

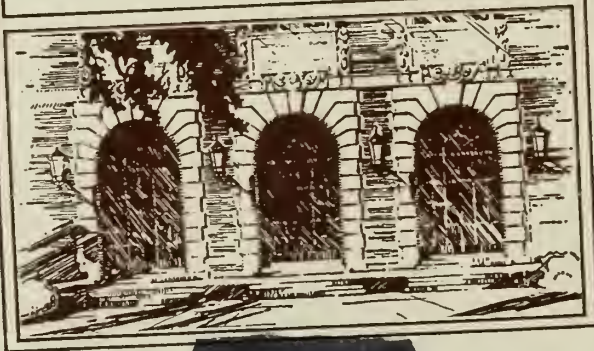
LIBRARY OF THE
UNIVERSITY OF ILLINOIS
AT URBANA-CHAMPAIGN

510.84

I l 6r

no. 308-315

cop. 2



The person charging this material is responsible for its return to the library from which it was withdrawn on or before the **Latest Date** stamped below.

Theft, mutilation, and underlining of books are reasons for disciplinary action and may result in dismissal from the University.

UNIVERSITY OF ILLINOIS LIBRARY AT URBANA-CHAMPAIGN

APR 26 1974

APR 9 REC'D



Digitized by the Internet Archive
in 2013

<http://archive.org/details/diskiofornoncore311bern>

0.81
E 262
20.311
Report No. 311

DISK I/O FOR NON-CORE-CONTAINED
P.D.E. MESHES AND ARRAYS

APR - 3 1969

by

Bruce Allen Bernott

March 5, 1969



DEPARTMENT OF COMPUTER SCIENCE
UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN · URBANA, ILLINOIS

THE LIBRARY OF THE
MAR 28 1969
UNIVERSITY OF ILLINOIS

Report No. 311

DISK I/O FOR NON-CORE-CONTAINED
P.D.E. MESHES AND ARRAYS*

by

Bruce Allen Bernott

March 5, 1969

Department of Computer Science
University of Illinois at Urbana-Champaign
Urbana, Illinois 61801

* This work was supported in part by the Advanced Research Projects Agency as administered by the Rome Air Development Center under Contract No. US AF 30(602)4144 and submitted in partial fulfillment of the requirements for the degree of Master of Science in Computer Science, February, 1969.

ACKNOWLEDGMENT

The author wishes to express his sincere appreciation to Professor Kuck for his efforts and guidance in the preparation of this report. Professor Kuck originally conceived the idea of CAT, a very high level language oriented toward solving partial differential equations numerically. (The language is also known by the name TRANQUIL II, as the origin of the acronym CAT seems to be shrouded in obscurity.) This report describes a part of what will be implemented in the compiler for that language; and with the advice of Professor Kuck, the author was able to keep the investigation relevant to the problem at hand.

Gratitude is also expressed to Mrs. Sharon Hardman, who has typed the final manuscript.

ABSTRACT

In finding discrete solutions of systems of partial differential equations on a computer, one is faced with the problem that the desired number of mesh points may exceed the machine's fast memory. This problem will be common on machines such as ILLIAC IV, for which extremely high computing power invites the use of meshes many millions of words in size. Because of the high dollar price of fast memory, it is sensible to look at large disk stores with high transmission speeds as back-up storage for meshes and arrays. The main problem encountered is the access time of such a storage unit.

The address of a block of data stored on disk might be taken as the address of the first word in the block. This address must specify both track number and radial position of the block. If the computer issues a command to transmit the block immediately after the beginning of the block has passed the reading head, then the system must wait for nearly a complete disk rotation before the transmission is started. This access time is, in general, not predictable; but it is bounded by the disk rotation time.

The access time during which the computer does useless work is latency; and the object of the present investigation to minimize the latency for a reasonably large class of problems.

TABLE OF CONTENTS

	Page
1. INTRODUCTION	1
1.1 <u>The Problem</u>	1
1.2 <u>The Machine</u>	4
2. MAPPING THE MESH ONTO DISK	6
3. SWEEPING A MESH.	10
3.1 <u>Normal Mode</u>	10
3.2 <u>Transposed Mode</u>	19
4. IMPLEMENTING THE SCHEME	25
4.1 <u>General</u>	25
4.2 <u>Measuring Latency</u>	32
4.3 <u>Storage Requirements</u>	34
4.3.1 <u>Fast Memory</u>	34
4.3.2 <u>Disk</u>	37
5. NUMERICAL SOLUTIONS.	40
6. CONCLUSION	49
APPENDIX	50
LIST OF REFERENCES	54

LIST OF TABLES

Table	Page
1. Positions of activity for row transition in row-normal reading	15
2. Acceptable values of ℓ for normal reading. "A" indicates acceptable; blank indicates unacceptable. $\ell = \ell_n$ or ℓ_m . Parentheses indicate value which is unacceptable for transposed reading.	17
3. Positions of activity for column transition k-2 to k-1 in column-transposed reading	23
4. Survey of results obtained on 11 mesh sizes. Latency $\leq .12$, normal reading in either direction	42

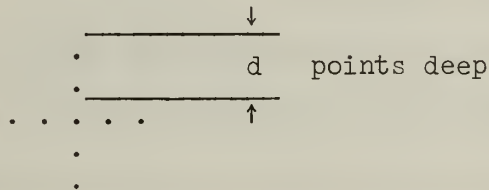
LIST OF FIGURES

Figure		Page
1.	Blocking a mesh	3
2.	Schematic of disk	5
3.	Grouping edge values for row-sweeping	7
4.	Mesh blocks and edge groups mapped onto a blocked disk . .	55
5.	Row-normal reading: row 2 and beginning of row 3. Entries enclosed in rectangles are written on disk; other entries are read from disk.	56
6.	Column-transposed reading	57
7.	Merging the end of column k-2 with the beginning of column k-1.	59
8.	Mapping edges into edge group blocks on disk.	28
9.	Illustrating relationships of T , T_R , and T_K	30
10.	T_R vs. N_V for the solutions found for four mesh sizes. The number of solutions is printed for each value of N_V . .	45
11.	Minimum values of T_R found	47

1. INTRODUCTION

1.1 The Problem

A reasonably large class of problems is the class of two-dimensional partial differential equation (PDE) problems. Characteristic of finite difference methods for solving PDE's are the stars or stencils of the methods, an example of which is shown below. In order to compute



new values of the variables (i.e., to update the variables) at the center point of the stencil, one needs to know the values of variables at several neighboring points in the horizontal and vertical directions. We may speak of the depth d of the stencil as the distance from the point to be updated to the furthest neighbor, measured in points. For the nine-point stencil illustrated above, the depth d is two points.

When one wishes to update an entire mesh of points, the stencil must be applied to each of the points simultaneously, so that neighbors used to update each point are old values and not updated values. If the entire mesh cannot be contained in the fast memory of the computer, one must store the mesh in a back-up storage device such as magnetic disk, and read in only a part of the mesh at a time for updating.

In this investigation, it is assumed that the mesh is rectangular in shape. It can then be sliced into rectangular blocks; and the blocks

may be transmitted back and forth between the disk and the computer memory for updating calculations. The object of this investigation is to formulate a scheme for efficiently swapping these blocks of mesh between disk and fast memory. One problem is that to update one block, the computer must have access to an edge of each of the neighboring blocks. The depth of this interblock communication must be d points, the depth of the stencil which is applied to the mesh. Figures 1a and 1b illustrate the slicing of the mesh and the communication necessary.

The program or subroutine which performs the updating calculations on a block and its neighbor edges can be the same for each block of mesh, with some conditional branches to handle blocks on mesh boundaries. This subroutine can be called the kernel, to distinguish it from the supervisory program which handles input/output and other chores. The main constituent of the kernel will be the stencil calculations of the method used.

There are two ways in which a kernel is likely to sweep or pass over the mesh: sequentially by rows or by columns. In sweeping by rows, blocks will be input (and output) in the order $11, 12, \dots, 1n, 21, 22, \dots, 2n, 31, \dots, mn$. In this case, it is clear that when updating block (i,j) , nothing special need be done to input edges from blocks $(i,j-1)$ and $(i,j+1)$. We simply save in fast memory the rightmost edge, d points deep, of block $(i,j-1)$, which will have just been updated and output to disk; and we delay calculations on (i,j) long enough to allow input of block $(i,j+1)$. The more difficult problem is arranging to have access to the lower edges of blocks in row $i+1$ and the upper edges of blocks in row $i-1$. The scheme presented in this report uses multiple storage of

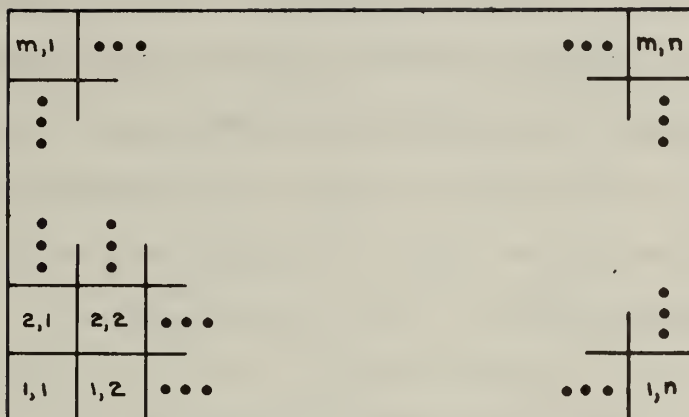


Figure 1a. A general rectangular mesh sliced into blocks.

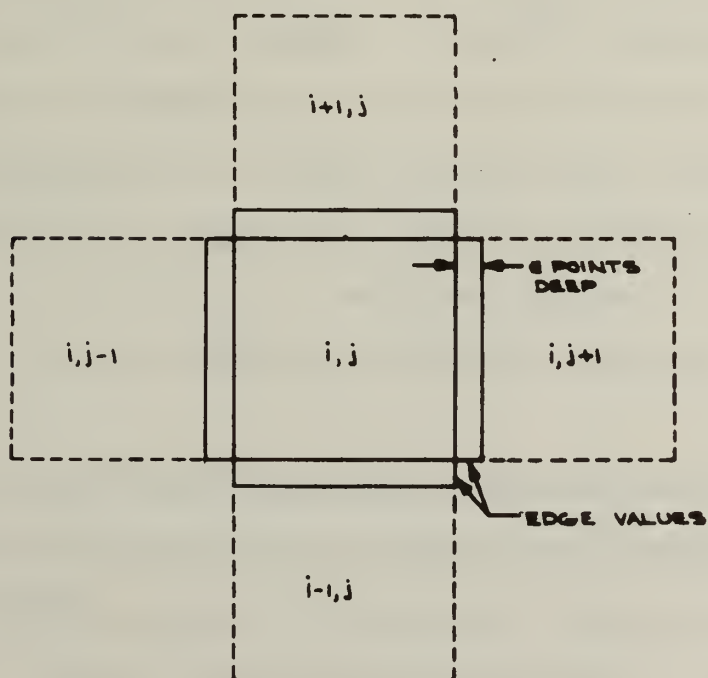


Figure 1b. Edge values needed for kernel calculations on block (i,j) .

upper and lower edge points on disk. Upper and lower edge points are grouped by rows into an "edge area" on disk, and also appear in the block storage or "mesh area". A similar arrangement is made for sweeping the mesh by columns; the essential difference between the two possible storage schemes is the grouping of edge values in the edge area. The transposition from row storage to column storage of edges and vice versa may be done during a sweep, so that successive kernels in a program may switch freely from row-sweeping to column-sweeping, if this is ever required.

1.2 The Machine

We will consider implementing an I/O scheme on one quadrant of ILLIAC IV with 2048 quadrant words, or 2048×64 words, of fast memory [1]. Coupled to the machine is DISK IV, which has a storage capacity of approximately 15 million words. DISK IV is organized into 48 tracks of 1200 segments each. Disregarding parity bits, each segment is equivalent to one long word across a four-quadrant ILLIAC IV, or four quadrant words. The smallest I/O transaction allowed is one segment. In the scheme to be proposed, a logical track will be divided into a number of blocks of b segments each. Each mesh block will be stored in some block of disk segments.

For a slight increase in generality, we will allow stringing several disk tracks together to form a logical track of $1200t$ segments, t an integer. We will have to take account of the fact that the read/write head cannot switch instantaneously from one track to another. DISK IV actually has a R/W head for every track, and switching is done electronically.

However, it still takes a measurable amount of time to change tracks. In fact, the switch can be performed within the space of disk revolution through two segments; thus two empty segments must appear at the end (or beginning) of each block of b segments and each 1200-segment disk track.

The disk may be pictured as in Figure 2. Points A and A' are the same radial position, so

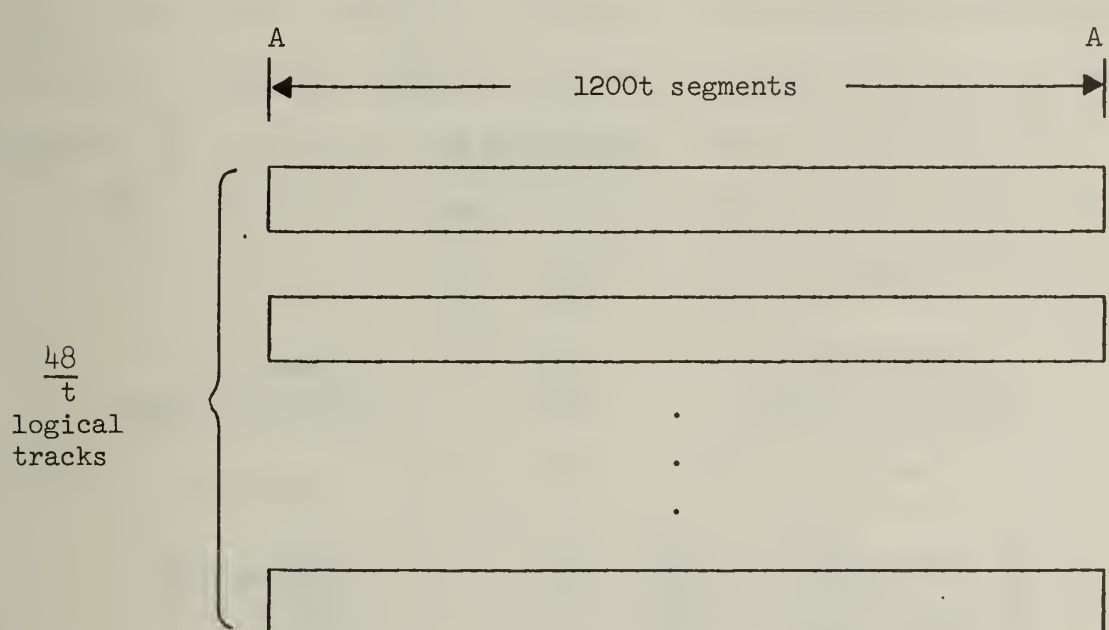


Figure 2. Schematic of disk.

that segments may be numbered modulo 1200t. The read/write head is imagined to move constantly from left to right, and can change tracks within the space of two segments.

2. MAPPING THE MESH ONTO DISK

The blocks (i,j) of Figure 1a are skewed on disk to allow row or column sweeping. For row sweeping upper and lower edge values from each row of blocks are duplicated into a number of logical tracks set aside for edge storage. These edge values are grouped as shown in Figure 3. Upper or lower edges from k blocks of mesh form one group of edges. The groups are numbered with Roman numerals to indicate row, superscripts U and L for upper and lower, and subscripts to distinguish groups in a particular row. For row R, group $\bar{R}_1^L$ contains edges from blocks $(R,1)$ to $(R,k-1)$, $\bar{R}_2^L$ contains edges from blocks (R,k) to $(R,2k-1)$, etc. Grouping for both $\bar{R}_1^L$ and $\bar{R}_1^U$ is summarized in the following table:

Group	Edges from Blocks in Columns:	Group	Edges from Blocks in Columns:
$\bar{R}_1^L$	1 to $k-1$	$\bar{R}_1^U$	1 to $k-2$
$\bar{R}_2^L$	k to $2k-1$	$\bar{R}_2^U$	$k-1$ to $2k-2$
.	.	.	.
.	.	.	.
.	.	.	.
$\bar{R}_{\ell_n}^L$	$(\ell_n-1)k$ to n	$\bar{R}_{\ell_n}^U$	$(\ell_n-1)k-1$ to n

The parameter ℓ_n is the smallest integer such that

$$n \leq \ell_n k - 2 \quad .$$

The overhanging outlines of edge groups at the mesh boundaries merely indicate that there are empty spaces in those edge groups. For sweeping the mesh by columns, left and right edges of blocks are grouped analogously. A smallest ℓ_m is chosen such that

$$m \leq \ell_m k - 2 \quad .$$

Note that k will not be subscripted. In problems requiring transposition, k will have the same value in both directions. All of the parameters used in constructing a scheme will be constant throughout problem execution.

For the purpose of describing the map of mesh onto disk, we will further simplify the schematic of disk by assuming the logical track to have a length of $kT-1$ (T defined below) blocks of b segments each. We will store each group of edges and each block of mesh in some block on disk as shown in Figure 4 (a fold-out chart). Figure 4 is a snapshot of a radial section of the disk. It is intended to appear as the left end of Figure 2. The map is drawn for "period" $T=5$ blocks. T measured in units of time will be the kernel calculation time for one block of mesh. T measured in blocks is the ratio of kernel calculation time for a block to the input transmission time for a block. In the analysis below, T will be measured in blocks.

Figure 4 shows edges grouped as in Figure 3; i.e., for row-sweeping. For column sweeping, edges would be regrouped and re-allocated, mesh blocks would retain the storage allocation shown. The reason mesh blocks need not be re-allocated is that skewing makes allocation look the same for both rows and columns if one considers radial positions only

(horizontally across page) and disregards track number (vertically down page). Block (i,j) is in the same radial position as block (j,i) . Let $p(X)$ be the radial position (disk block number measured from an arbitrary radial point A) of record X, where X is either an edge group or a mesh block. The positions of all records are defined in terms of $p(I_1^U)$.

$$p(\overline{R+1}_1^{U/L}) = p(\overline{R}_1^{U/L}) + T$$

$$p(\overline{R}_1^L) = p(\overline{R}_1^U) + T$$

$$p(\overline{R}_{i+1}^{U/L}) = p(\overline{R}_i^{U/L}) + 1$$

$$p((1,1)) = p(III_1^U) + 2$$

$$p((i,j+1)) = p((i+1,j)) = p((i,j)) + T$$

Besides these, we add the requirement

$$p((i,j+k)) = p((i,j)) + 1 \quad (=p((i+k,j)))$$

which relates the grouping constant k to the length of the logical track. It defines the logical track to be of length $(kT-1)$ blocks.

Implementation of the simplified structure of disk used in Figure 4 on an actual disk with 1200-segment tracks will be taken up in a later section. For now we will assume zero head-switching time, and proceed to describe the sequence of I/O transactions for sweeping the mesh.

3. SWEEPING A MESH

3.1 Normal Mode

If we are to sweep the mesh by rows, we wish to read into fast memory the blocks $(1,1)$, $(1,2)$, $(1,3)$, ..., $(1,n)$, $(2,1)$, $(2,2)$, ... in sequence. Ignoring edge value transmissions for the time being, we read block $(1,1)$ from logical track i in Figure 4 as the read/write head moves from left to right. Let calculations on block $(1,1)$ begin as the R/W head passes the point $p((1,2))-1$ (point $p(X)$ is the beginning of position $p(X)$). While calculations proceed on block $(1,1)$, block $(1,2)$ is read. Since we intend to read each block of row 1 of mesh as it passes by, the kernel must finish the calculations on each block within the period T . We will write the updated block $(1,1)$ in position $p((1,3))-1$ (on another logical track if necessary). At the point $p((1,3))-1$ we begin calculations on block $(1,2)$. Immediately after writing the new block $(1,1)$ we read block $(1,3)$. In general, we read block (i,j) , write updated block $(i,j-1)$, read $(i,j+1)$, write (i,j) , etc. in sequence. There will, of course, be a "hiccup" at the end of each mesh row, except in one case to be discussed later. If necessary, we spend a whole or part of a disk rotation of latency to insure that the finishing of one row does not interfere with the beginning of the next row.

After sweeping the whole mesh in this way, the updated mesh will be stored on disk in the same configuration, but shifted $2T-1$ blocks to the right. Column-sweeping (i.e., $(1,1)$, $(2,1)$, $(3,1)$, ..., $(m,1)$, $(1,2)$, $(2,2)$, ...) results in the same shift. It is evident that we may sweep by rows or columns any number of times in any order.

We now superimpose edge group transmissions upon the I/O sequence just described. The term "reading normal" will be applied to sweeping the mesh by rows with edge values grouped for row-sweeping (as in Figures 3 and 4) and to sweeping by columns with edge values grouped for column-sweeping. The other two possibilities are column-sweeping with row-grouped edges and row-sweeping with column-grouped edges; the term "reading transposed" is applied to these types of sweeps.

In reading normal by rows, edge values to the left and right of a block (as viewed in Figure 3) are input automatically because of the sequence of block reading, as mentioned before. Similarly for reading normal by columns, edge values below and above a block are input automatically. Edge values below and above for row-normal reading and left and right for column-normal reading will be input from the edge area on disk. The procedure for row-normal reading will be shown. Column-normal reading is analogous: edge group superscript U (upper) is replaced by R for "right" and L (lower) is replaced by L for "left".

Row-normal reading is shown in Figure 5. This chart is not a storage map as in Figure 4. Figure 5 is read a line at a time from left to right. Each numbered line represents one logical disk revolution. Each entry on a line represents the transmission of a record to or from the disk. Entries in rectangles are written on disk, other entries are read from disk. Horizontal position corresponds to radial position on a logical track exactly as in Figure 4. Each read entry is, of course, in the position indicated by Figure 4. It may be helpful to align Figure 5 on Figure 4.

Only the I/O sequence for one row of mesh blocks need be described since the relationship between positions of edge groups required for that row and positions of blocks in that row are independent of row number. We will follow through the sequence for row 2 since row 1 is a slightly degenerate case.

On revolution 1 we must initialize the calculation by reading the first upper edge group of row 1 I_1^U and the first lower edge group of row 3 III_1^L . With this data in fast memory we may read and update the first $k-2$ blocks of row 2, since I_1^U contains edges from the 1st $k-2$ blocks. We, therefore, start reading the blocks sequentially from (2,1) on revolution 1 and write each block, updated, $2T-1$ disk blocks after it is read. This eventually brings us back to radial point A where revolution 2 begins. We read edge group I_2^U , which contains edges below blocks (2, $k-1$) to (2, $2k-2$), as it goes by. Notice that we have read the edge below (2, $k-1$) well before we actually need it. Later on in revolution 2 we read III_2^L which contains edges above blocks (2, k) to (2, $2k-1$). The edge above block (2, k) is read in at $p((2,k)) + T-2$; but this is acceptable since calculations on (2, k) are not started until the point $p((2,k)) + T-1$.

Besides writing updated blocks as we move along the disk, we must write updated edge groups. These groups must be written in positions such that the disk configuration of the updated mesh and edge groups is the same as Figure 4. Hence, we must write the updated edge group II_1^U $2T+2$ disk blocks before the position of the updated block (2,1), and II_1^L $T+2$ disk blocks before updated block (2,1). Fortunately, so to speak, there are no other activities which must occupy the R/W head in

these positions on revolution 2^* ; and the fact that $II_1^{U'}$ and $II_1^{L'}$ must contain updated edges from blocks $(2, k-2)$ and $(2, k-1)$ respectively is no problem, since those blocks will have been updated by the time the corresponding edge groups need be written. There must, of course, be buffers in fast memory for accumulating updated edge groups prior to transmissions to disk.

On revolution 3 the same procedure is followed with all transmissions shifted to the right by one block, all subscripts incremented by 1 and all j -values of blocks (i, j) incremented by k . With each revolution transmissions shift right one disk block by the fact that we have arranged the logical track to be $kT-1$ blocks long and mesh block transmissions have period T .

Earlier we defined ℓ_n such that

$$(\ell_n - 1)k - 2 < n \leq \ell_n k - 2 \quad .$$

Let us say that $n = \ell_n k - 2$. Then the last transmissions for row 2 of mesh appear as shown on revolution $\ell_n + 1$ in Figure 5. If $n < \ell_n k - 2$, then some mesh block transmissions disappear, but updated edge groups $II_{\ell_n}^{U'}$ and $II_{\ell_n}^{L'}$ must still be written in the positions shown. Processing of row 3 on some revolution r_3 may begin after row 2 is finished. We may have $r_3 = \ell_n + 2$; however, in this case we spend more than a revolution of

* We do not attempt to take advantage of the machine's capability to perform two transmissions simultaneously. This capability arises from having two electronics units, each with half of the total number of disk storage units. As a result, any simultaneous transmissions must occur in opposite halves. We do not attempt to meet this restriction.

latency ($kT-1 + p((3,1)) - (p((2, \ell_n^{k-2})) + T)$ disk blocks). We would like to have $r_3 = \ell_n + 1$. This cannot be done with the sequence shown in Figure 5 because of conflicts in $p(II_1^U)$ and $p((3,1))$. The positions of activity on revolution $\ell_n + 1$ are for $\ell_n = T-1$ with $T = 5$. If ℓ_n were 5, line $\ell_n + 1$ would be shifted one block right, and we could have $r_3 = \ell_n + 1$. It is clear that the value of ℓ_n is critical in determining latency.

Acceptable values of ℓ_n , i.e., those for which r_3 may equal $\ell_n + 1$, can be found in general by comparing activity in revolution r_3 to activity in revolution $\ell_n + 1$. Take $p(II_1^U) = 0$. Then revolution r_3 has activity in positions 0, $3T$, $3T+2$, $4T+2$, $5T+1$, $5T+2$, ... We also find positions of activity for revolution $\ell_n + 1$ by first determining $p((2, \ell_n^{k-2}))$:

$$p((2,1)) = 2T + 2$$

$$p((2,k)) = p((2,1)) + 1 - T$$

$$p((2,k-2)) = p((2,k)) - 2T$$

$$p((2, \ell_n^{k-2})) = p((2,k-2)) + (\ell_n - 1).$$

Therefore, $p((2, \ell_n^{k-2})) = 2 + \ell_n - T$. Other activity in revolution $\ell_n + 1$ can be determined from this fix. Positions of activity are summarized in Table 1.

<u>ENDING ROW 2</u>		<u>STARTING ROW 3</u>	
<u>ACTIVITY</u>	<u>POSITION</u>	<u>ACTIVITY</u>	<u>POSITION</u>
$(2, l_n^k - 4)$	$2 + l_n - 3T$	II_1^U	0
$(2, l_n^k - 5)$	$1 + l_n - 2T$	IV_1^L	$3T$
$(2, l_n^k - 3)$	$2 + l_n - 2T$	$(3,1)$	$3T + 2$
$(2, l_n^k - 4)$	$1 + l_n - T$	$(3,2)$	$4T + 2$
$(2, l_n^k - 2)$	$2 + l_n - T$	$(3,1)$	$5T + 1$
$(2, l_n^k - 3)$	$1 + l_n$	$(3,3)$	$5T + 2$
$(2, l_n^k - 2)$	$1 + l_n + T$		
$II_{l_n}^U$	$-2 + l_n + 2T$		
$II_{l_n}^L$	$-2 + l_n + 3T$		

Table 1. Positions of activity for row transition in row-normal reading.

We will not take into account any activity before $p((2, \ell_n^k - 4))$ in ending row 2 or after $p((3,3))$ in beginning row 5. This will lead to an analysis which is correct for $2 + \ell_n - 3T \leq 0$ and $-2 + \ell_n + 3T \leq 5T + 2$; or $\ell_n \leq 3T - 2$ and $\ell_n \leq 2T + 4$. It has not yet been mentioned that the scheme proposed will work only for $T \geq 5$. We look for acceptable values of $\ell_n \leq 10$, which satisfies both inequalities above for $T \geq 5$. A value is acceptable if the two sets of positions in Table 1 are disjoint. The sets are disjoint for $\ell_n = T$, for example. $\ell_n = T-3$ is unacceptable for $T = 5, 7$, acceptable otherwise ($T \geq 5$). $\ell_n = T-2$ is always unacceptable.

In general, a value can be tested for acceptability by substituting that value into the left column and comparing each value obtained with all of the elements in the right column, for all values of T . Whether one substitutes numbers or expressions in T for ℓ_n , it is a painful task. Table 2 shows acceptable numerical values of ℓ_n versus T . For any numerical value of ℓ_n , a minimum K_{ℓ_n} can be found such that for $T \geq K_{\ell_n}$ the ℓ_n is either always acceptable or always unacceptable. $K_{10} = 13$: $\ell_n = 10$ is acceptable for $T \geq 13$. $K_i < K_{10}$, $i = 1, \dots, 9$. The table is therefore not shown for $T > 13$. Table 2, along with a corresponding table for transposed reading, will be useful in finding a scheme with minimum latency for a given problem.

$\begin{array}{c} T \\ \ell \end{array}$	5	6	7	8	9	10	11	12	13
1	A	A	A	A	A	A	A	A	A
2									
3		A	A	A	A	A	A	A	A
4									
5	A			A	A	A	A	A	A
6	A	A			A	A	A	A	A
7		A	A			A	A	A	A
8			A	A			A	A	A
9		A		A	A			A	A
10	(A)		A		A	A			A

Table 2. Acceptable values of ℓ for normal reading. "A" indicates acceptable; blank indicates unacceptable. $\ell = \ell_n$ or ℓ_m . Parentheses indicate value which is unacceptable for transposed reading.

Only row-normal reading has been considered in this section; but it is clear that all of the results apply also to column-normal reading with column parameters replacing corresponding row parameters--namely, ℓ_m , m , and edge group superscripts R and L . As mentioned before, k is the same in both directions.

We now examine the problem of changing over from sweeping by rows to sweeping by columns. Note in Figure 5 that when updated edge groups $II_i^{U'}$ and $II_i^{L'}$ were written on disk, the upper and lower edges of groups of k blocks in a row were written. This prepares the disk storage structure for a subsequent sweep by rows. If the next sweep is to be by columns, however, we store on disk the updated right and left edges of the blocks in the row rather than the upper and lower edges respectively. We will use the notation $II_i^{R'T}$ and $II_i^{L'T}$. $II_i^{R'T}$ is written in place of $II_i^{U'}$, and $II_i^{L'T}$ is written in place of $II_i^{L'}$. The problem arises that these right and left edges are still grouped by rows; i.e., $II_i^{R'T}$ contains updated right edges from blocks $(2, (i-1)k - 1)$ to $(2, ik - 2)$, and $II_i^{L'T}$ contains updated left edges from blocks $(2, (i-1)k)$ to $(2, ik - 1)$.^{*} Edges in all of the $\bar{R}_i^{R'T}$ and $\bar{R}_i^{L'T}$ are grouped in the same way. Since these edges are not grouped for column-normal reading, a different I/O sequence will be required. This new sequence will be called column-transposed reading. Instead of reading k edges from one place in a logical disk revolution, we will read the edges one at a time from many points in a

^{*}It may be helpful to imagine the individual edges (k per edge group) in Figure 3 to be rotated clockwise by ninety degrees.

revolution, so that the flow of edge values into fast memory becomes a quasi-continuous process as opposed to the single transmissions in Figure 5.

3.2 Transposed Mode

Figure 6,a and b, illustrates the procedure for column-transposed reading. The mesh is assumed to be mapped on disk as in Figure 4; except that upper edge groups $\bar{R}_i^U$ have been replaced with right transposed edge groups $\bar{R}_i^{RT}$ and lower edge groups $\bar{R}_i^L$ have been replaced with left transposed edge groups $\bar{R}_i^{LT}$.

A sweep on column 2 will be used as a first example. The first block to be input is (1,2), for which we need the right edge of (1,1) and the left edge of (1,3). The right edge of (1,1) is contained in edge group I_1^{RT} ; specifically, it is the third edge in I_1^{RT} , counting the two empty edge spaces overlapping the mesh in Figure 3. Likewise, the left edge of (1,3) is the fourth edge in I_1^{LT} . The edges needed for block (2,2) are the third and fourth edges in edge groups II_1^{RT} and II_1^{LT} respectively. In order to move up column 2, we must have one edge each from all edge groups with subscript one. Figure 6a shows "simultaneous" transmissions of single edges from pairs of edge groups. An edge from II_1^{RT} and an edge from I_1^{LT} , for example, are to be read within the space of one disk block. The reading head will be required to switch tracks within a disk block, and the edges within the groups involved must be arranged such that no pair of edges required in any transmission have the same radial position. This matter will be taken up in a later section; for now we assume the requirement to be satisfied.

As each successive column is swept, the beginning of the mesh block transmissions moves T disk blocks to the right because of skewing. The edge value transmissions, however, start at the same radial position up to and including the sweep on column $k-2$. We are reading edges farther and farther ahead of corresponding mesh blocks. When mesh blocks have finally shifted around an entire logical track, we will be unnecessarily reading edges a disk revolution ahead of the mesh blocks for which the edges are needed. At this point we may shift to reading edges immediately before corresponding blocks. Such a shift is executed in two stages in Figure 6b.

The first stage of the shift is executed at column $k-1$. Only the transmissions of left edges from mesh blocks in column k are shifted. Transmissions of right edges of column $k-2$ are still started a revolution ahead of mesh block transmissions. Column $k-1$ is a special case, as its eastern neighbor edges are now in edge groups with subscript 2 while its western neighbor edges are still in edge groups with subscript 1. It would be possible to shift transmissions of both left and right edges at column $k-1$; however, shifting the left edges only results in a greater freedom with the parameter ℓ_m , when one considers the efficiency of the transition from column $k-2$ to column $k-1$.

The second stage of the shift is executed at column k . Now all edges needed are in edge groups with subscript 2. The sweep up column $k+1$ is similar to the sweep up column 1, except that column 1 has no western neighbor edges. Likewise, the sweep up column $k+2$ is similar to the sweep up column 2; and in general the sweep up column j is similar to the sweep

up column j modulo k . Columns $nk-1, n=1,2,\dots,\ell_n-1$, all require edges from edge groups with different subscripts.

When the updated edge values are written, they will, of course, be grouped in the column direction. If left and right edge groups $\bar{R}_i^{L'}$ and $\bar{R}_i^{R'}$, $\bar{R} = I, II, \dots, \bar{n}$, are written, then the updated mesh will be organized for column-normal reading. If the subsequent sweep were to be by rows, lower-transposed $\bar{R}_i^{L'T}$ and upper transposed $\bar{R}_i^{U'T}$ would be written instead. If R indicates a sweep by rows and C a sweep by columns, then sequences of the type $(RC)^n$ will have every read in the transposed mode. Sequences of the type $(RRCC)^n$ will have alternating normal and transposed reading sequences.

Merging the transmissions at the end of one column with the transmissions at the beginning of the next column is slightly more complex than for reading normal in that edge transmissions at the beginning of a column may penetrate very deeply into the transmissions for the preceding column. The transitions from column $k-2$ to column $k-1$ and from column $k-1$ to column k are worst cases. Both of these transitions will be inspected.

Figure 7 illustrates the ending of column $k-2$ and the beginning of column $k-1$. Let the reference position $p((I_1^{RT}))$ be zero. For starting column $k-1$, there is activity in positions $0, T, 2T, \dots, p((\overline{k+1}_1^{RT})) = kT \bmod(kT-1) = 1, p((1,k-1)) = 3, 1+T, 3+T, 1+2T, 2+2T, 3+2T = p((3,k-1))$. Positions of activity for ending column $k-2$ may be computed by first obtaining a fix on block $((\ell_m-1)k+1,k-2)$:

$$p((1,k-2)) = p((1,k-1)) - T = 3 - T$$

$$p(((\ell_m-1)k+1,k-2)) = p((1,k-2)) + (\ell_m-1) = 2 + \ell_m - T.$$

We must compare revolutions A with C and B with D. Note that for the first comparison, we need only compare activity between markers M_1 and M_2 on A with the first activity on C because of the periodicity T of activities and the facts that $p(\boxed{(\ell_m - 1)k, k-2}) = 1 + \ell_m > p((I_1^{RT}))$ and $p(\boxed{\ell_m k - 2, k-2}) = \ell_m - 2T \leq p((I_1^{RT}))$ for $\ell_m \leq 2T$. We will again look for values $\ell_m \leq 10$.

For the comparison of B with D, note that $p(\boxed{\ell_m k - 2, k-2}) = 2 + \ell_m - 2T < p((1, k-1)) = 3$ for $\ell_m \leq 2T$; so that we need only take account of possible conflicts between the two updated edge group transmissions on B and the mesh block transmissions on D. Transmissions to the left on B and D will be compatible if transmissions to the right on A and C are compatible. Also the two updated edge group transmissions on B will not conflict with edge transmissions on D if updated edge group transmissions on A do not conflict with (I_1^{RT}) for $\ell_m \leq 2T$. Of the mesh block transmissions on D we need only consider $(1, k-1)$ and $(2, k-1)$ since the next one, $\boxed{1, k-1}$, is always to the right of the last activity of B for $\ell_m \leq 2T$. Actually, we need only consider the last transmission on B.

Table 3 lists the activities of interest in making the column transition. Sets A and C must be disjoint and sets B and D must be disjoint. If we construct a table of acceptable values such as Table 2, we find that there is only one unacceptable value pair (ℓ_m, T) which is not on Table 2. That pair is $\ell_m = 10, T = 5$. If we delete the corresponding "A" from Table 2, we have a table of acceptable values for both normal and transposed reading.

<u>ENDING COLUMN k-2</u>		<u>STARTING COLUMN k-1</u>	
<u>ACTIVITY</u>	<u>POSITION</u>	<u>ACTIVITY</u>	<u>POSITION</u>
<u>A</u>		<u>C</u>	
$(\overline{\ell_m^{k-2}}_1^{LT})$	ℓ_m^{-2T}	(I_1^{RT})	0
$\boxed{(\ell_m - 1)k - 2, \quad k - 2}$	$1 + \ell_m^{-2T}$		
$((\ell_m - 1)k, k - 2)$	$2 + \ell_m^{-2T}$		
$\boxed{\overline{k-2} \ell_m^{-1} R'}$	$-2 + \ell_m^{-T}$		
$\boxed{(\ell_m - 1)k - 1, \quad k - 2}$	$1 + \ell_m^{-T}$		
$((\ell_m - 1)k + 1, k - 2)$	$2 + \ell_m^{-T}$		
$\boxed{\overline{k-2} \ell_m^{-1} L'}$	$-2 + \ell_m$		
<u>B</u>		<u>D</u>	
$\boxed{k - 2 \ell_m^{L'}}$	$-1 + \ell_m$	$(1, k - 1)$	3
		$(2, k - 1)$	$3 + T$

Table 3. Positions of activity for column transition k-2 to k-1 in column-transposed reading.

One might question the statement that we have obtained all unacceptable pairs (ℓ, T) for transposed reading since we have considered only one transition out of k different transitions. In fact, it can be verified that we have all unacceptable values for $\ell_m, \ell_n \leq 10$ and $T \geq 5$.

Having investigated most of the logical aspects of the scheme under consideration, we now consider the problem of implementing the scheme on a disk storage unit with 1200 addressable segments per track.

4. IMPLEMENTING THE SCHEME

4.1 General

For mesh storage in one quadrant of ILLIAC IV, it is efficient to store 8×8 squares of mesh points in "quadrant words" across the 64 processing elements. For this reason, it will be assumed that the mesh is subdivided into 8×8 squares; and mesh blocks will have dimensions of $p \times q$ squares, p and q integers. The smallest addressable piece of data on the disk is the segment, which consists of 256 words, or 4 quadrant words. The head-switching time of the disk, again, is taken as two segments.

We construct each logical track pictured in Figure 4 from t disk tracks for some integer t . In doing so, we do not take advantage of the $t-1$ times an actual radial position is passed within each logical revolution. If b is the number of segments in a disk block, then

$$b(kT-1) \leq 1200t$$

must be satisfied. We might try

$$b = \left\lfloor \frac{1200t}{kT-1} \right\rfloor \quad \text{segments,}$$

where the operator $\lfloor \rfloor$ indicates the greatest integer less than or equal to the argument. The truncation will result in wasting a number of segments; we allow this if the number of unused segments is reasonably small. These wasted segments become a dead area on disk and will never be used for storage. The dead area will, of course, contribute to overall latency.

Of the b segments in a disk block, we may use $b-2$ segments, with 2 segments at the end or beginning of the block for head switching. If b

does not divide 1200, then some disk blocks may straddle track connections and we will, in general, have to allow 2-segment spaces at each of the $t-1$ connections of the disk tracks. If the dead area is not large enough to cover this, one might reduce b or try another combination of k and t .

In an actual mesh there will be a number of variables associated with each mesh point. Let this number be N_v . We must have

$$N_v p q \leq 4(b-2) \quad .$$

In addition, if the mesh has dimensions $M \times N$ 8×8 squares,

$$M \leq p m$$

and

$$N \leq q n$$

where m, n are dimensions in mesh blocks.

The ℓ_m, ℓ_n satisfy

$$(\ell_m - 1)k - 2 < m \leq \ell_m k - 2$$

$$(\ell_n - 1)k - 2 < n \leq \ell_n k - 2$$

and they should be acceptable as defined by Table 2.

Noting that there are k edges per edge group, one edge must be contained in

$$\left\lfloor \frac{b-2}{k} \right\rfloor \quad \text{segments}$$

for the reason that individual edges must be addressable for transposed reading. An edge consists of $N_v d(8p)$ or $N_v d(8q)$ points. Hence

$$8N_v d \text{ Max}(p, q) \leq 256 \left\lfloor \frac{b-2}{k} \right\rfloor \quad .$$

The number of segments required for an edge is

$$\left\lfloor \frac{8N_v d \text{Max}(p,q) + 255}{256} \right\rfloor .$$

If it is not necessary to change the direction of sweeping over the mesh, then edges need not be individually addressable, and the requirement is less stringent:

$$8N_v d s k \leq 256(b-2)$$

where s is p or q depending on the direction of sweeping.

In the description of transposed reading, it was assumed that when two edges from different edge groups were required within the space of one disk block, the two edges would not be in the same radial position. In fact, the two edges must be separated by at least two segments. For $k \geq 6$ this can be guaranteed as follows.

Observe from Figure 8 that for sweeping up column $(i-1)k+j-2$ we must read the $(j-1)$ th edge from a superscript RT edge group and the j th (modulo k) edge from a superscript LT edge group. Each edge is mapped into an "edge slot" on a logical track as shown. All LT edge groups begin with the $(k-1)$ th edge in the first edge slot; and all RT edge groups begin with the 1st edge in the first edge slot. Then the edges required for transmission are separated by two edge slots for $k \geq 6$. The smallest edge slot possible is one segment. In moving up column $ik-1$, we need the k th edge from $\overline{R+k+1}_1^{RT}$ and the 1st edge from $\overline{R}_{i+1}^{LT}$, for which the requirement is still satisfied. If the edge slot is two segments or more, then $k \geq 4$

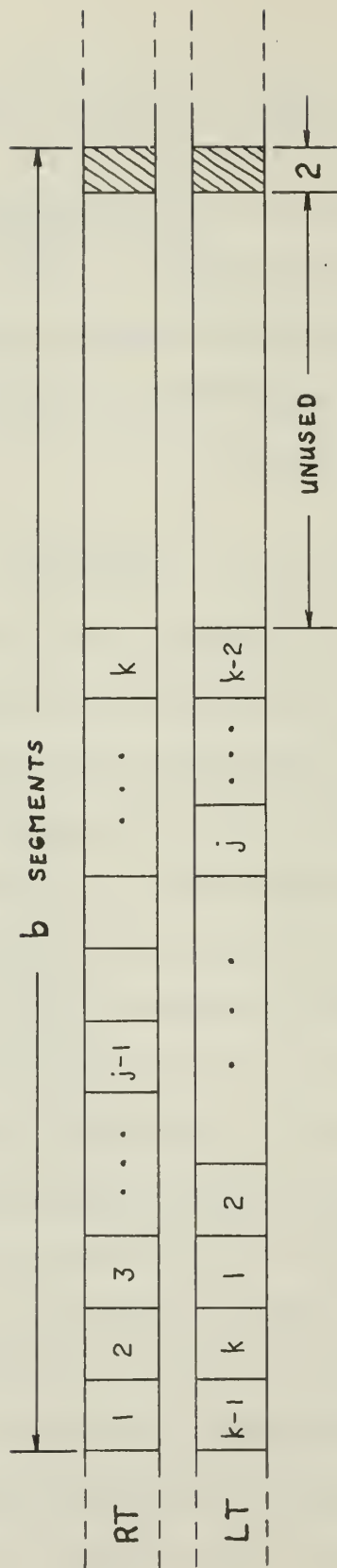
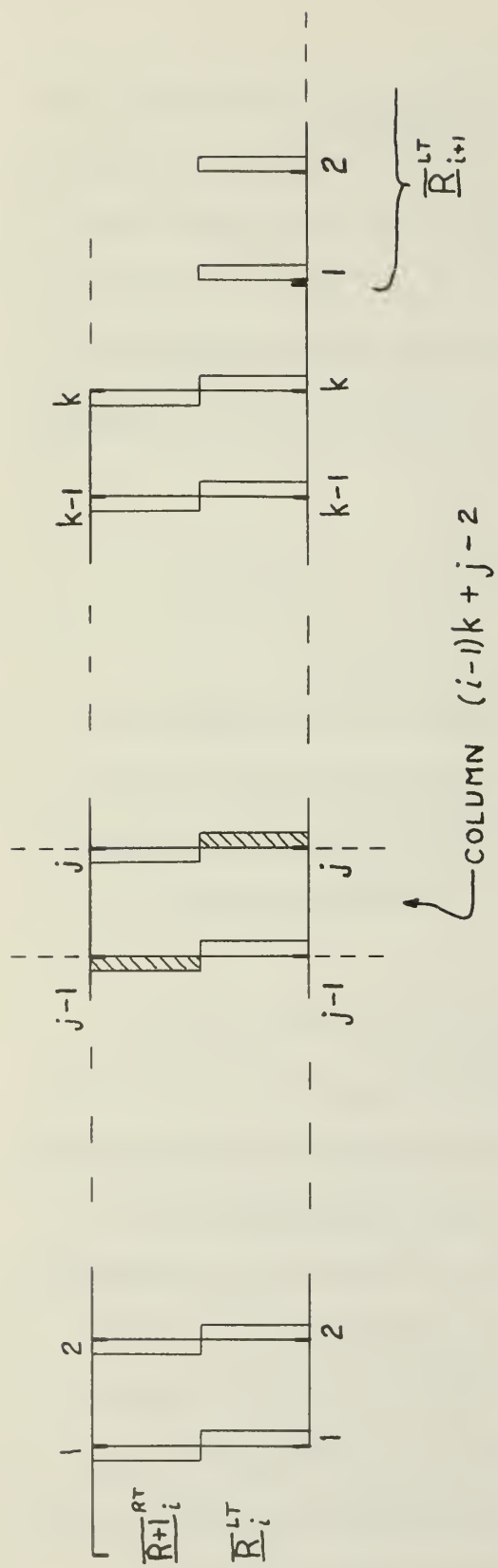


Figure 8. Mapping edges into edge group blocks on disk.

is sufficient. Of course, the unused portion of the $b-2$ segments allotted might be distributed between edge slots to achieve separation.

No mention has yet been made of having T be some value ≥ 5 other than an integer. This can certainly be done without upsetting the logic of the I/O sequence. Another point concerning T should be considered, however.

Recall that T is the ratio of the allowed compute time for a mesh block to the one-way transmission time for a disk block of b segments. T is the logical period of the scheme, and it is a parameter of great interest to us. When dealing with a calculation kernel, however, we are more interested in the ratio of the allowed compute time to the input time for a mesh block, which is always less than a disk block by 2 segments or more. Let this ratio, the real period, be devoted by T_R . $N_{vpq}/4$ segments are used to store a mesh block; therefore

$$T_R = \frac{4Tb}{N_{vpq}} \quad .$$

Figure 9 presents the relationship graphically.

Both T and T_R are measures of the same quantity, the amount of time available for computation on a mesh block. They are measures in different units, however, as indicated in Figure 9. We are more interested in T_R than in T because kernel times can be given in mesh blocks. By mesh block units of time, we mean, of course, the transmission time for a mesh block of $N_{vpq}/4$ segments. Although a mesh block is actually transmitted in $\lceil (N_{vpq}+3)/4 \rceil$ segments, we are interested in the time to transmit that

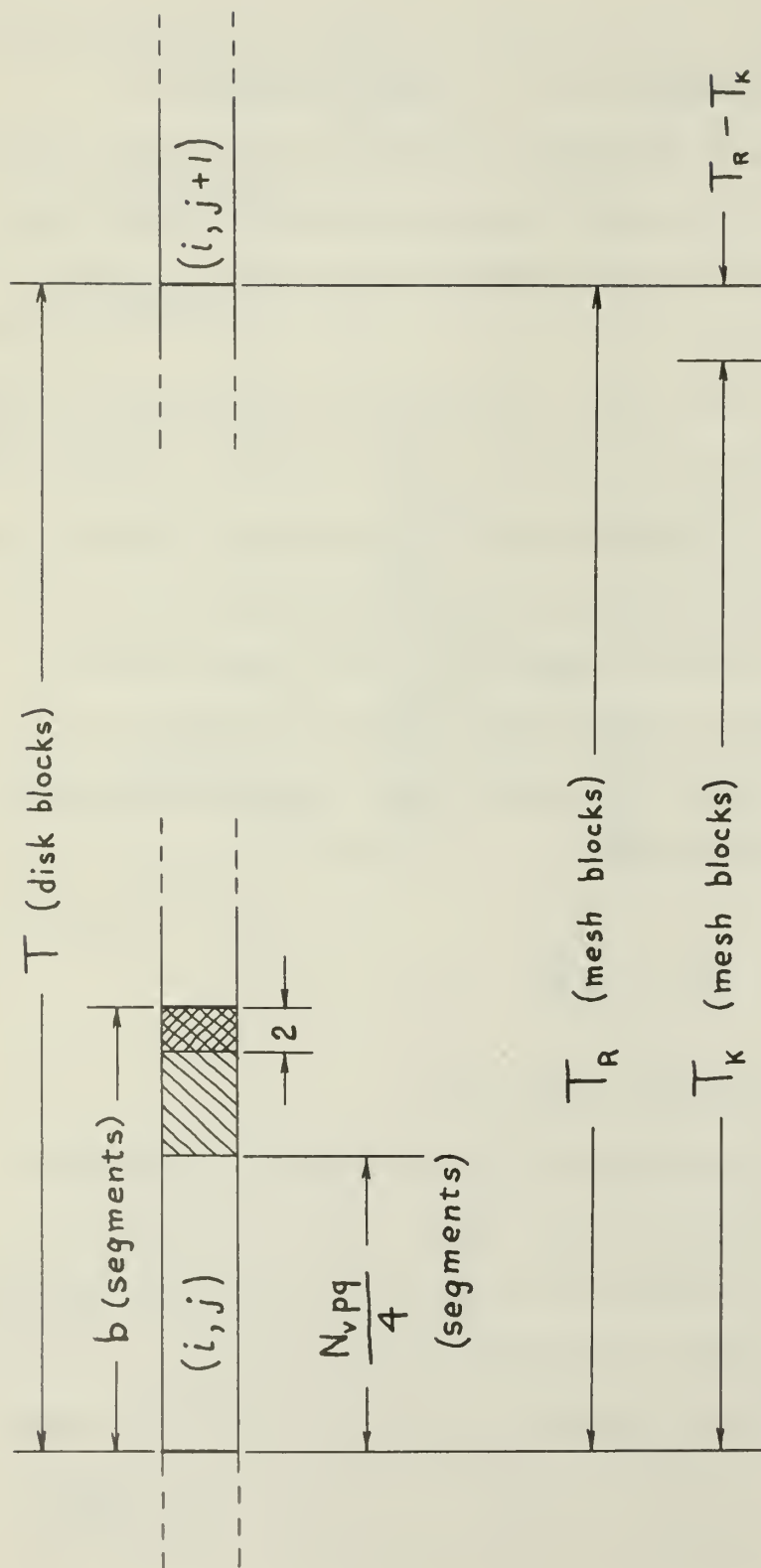


Figure 9. Illustrating relationships of T , T_R , and T_K .
 Disk block $\equiv b$ segments.
 Mesh block $\equiv N_v pq/4$ segments.

part of $\lfloor (N_{vp}+3)/4 \rfloor$ segments containing data. It is obvious that T is a lower bound on T_R and that T_R cannot equal T because of the two segments reserved for head switching.

With every kernel there can be associated a number T_K which is the ratio of the calculation time per point to the input time per point. T_K is the calculation time per point normalized to the disk transmission rate; it is the kernel time in mesh blocks. It is assumed that T_K is independent of the size of the mesh block which the kernel is updating; although if the compute time varies slightly because of I/O interrupts, etc., T_K should be an upper bound on the compute time. We must have $T_K \leq T_R$; and we wish to have T_R as close to T_K as possible in order to match the overall transmission rate of the scheme to the speed of the kernel. If several different kernels are applied to the mesh, then we match the scheme parameter T_R to the slowest kernel, since it appears that the parameters of the scheme cannot be changed during a sweep.

Given a mesh of dimensions $M \times N$ 8×8 squares with N_v variables per point, a stencil depth d , and a normalized kernel time T_K , the problem is to find scheme parameters for which the latency of the disk and the value $T_R - T_K$ are as small as possible. In reality, $T_R - T_K$ is part of the overall latency since for $T_R > T_K$ the data on disk is "not there" exactly when we are ready to compute on it. The value $T_R - T_K$ can be thought of as the ratio to T_b segments, of the number of segments for which the computer is waiting for the disk to send another mesh block. In spite of this, the present discussion will distinguish between overall latency and the value $T_R - T_K$.

4.2 Measuring Latency

The measure $T_R - T_K$ represents latency which, in a sense, can be made to go away by increasing T_K . It is not proposed to increase the complexity of a given kernel for no reason; but rather if a solution of the scheme parameter relations exists, then one might look for a problem with T_K close to the T_R for the scheme. $T_R - T_K$ might be called external latency.

The question now is: "What is internal latency?" We can include the wasted time between complete sweeps of the mesh, the "hiccups" between sweeps along rows or columns of the mesh, latency due to incompletely-filled blocks on the last row or column, and the time spent skipping over the dead area on disk. The last three elements will be discussed here.

The size of the dead area on each logical track is $1200t-b(kT-1)$ segments. Since it is passed on each logical revolution, the percent of total time spent on the dead area, i.e., the dead area latency, is

$$L_D = \frac{1200t-b(kT-1)}{1200t} \quad .$$

It is unlikely that any code could do useful work to mask the dead area latency since the dead area is not distributed across the disk blocks in a logical track, and furthermore since it moves left by $2T-1$ disk blocks relative to the mesh with each complete sweep.

The "hiccups" between sweeps along rows and columns will cause latency unless $n = \ell_n^{k-2}$ and $\ell_n = 3T$, and likewise for m . For $n = \ell_n^{k-2}$ there will be $3T - \ell_n$ disk blocks of latency at the end of each row. In addition, if $\frac{N}{q} \leq n \leq \ell_n^{k-2}$, we may not be using all of the ℓ_n^{k-2} mesh blocks allowed. Then $\ell_n^{k-2} - \frac{N}{q}$ blocks are effectively wasted. For every

block so wasted we incur an additional T blocks of latency. This occurs at the end of every row, out of disk revolution through approximately $\ell_n(1200t) + Tb$ segments. The latency incurred at row connections is therefore

$$L_R = \frac{(3T - \ell_n + T(\ell_n^{k-2} - \frac{N}{q}))b}{\ell_n(1200t) + Tb} \quad .$$

Similarly, for column sweeping,

$$L_C = \frac{(3T - \ell_m + T(\ell_m^{k-2} - \frac{M}{p}))b}{\ell_m(1200t) + Tb} \quad .$$

Additional latency occurs on the last row for row-sweeping if $M < pm$. The dimension m should be the smallest integer such that $M \leq pm$. This latency, like the latency at row connections, might be masked by useful calculation on the boundaries of the mesh. Extra computing time can be provided for all four boundaries by strictly enclosing the $M \times N$ mesh in the $m \times n$ blocks. It will be assumed here, however, that updating calculations on boundary points are no more time-consuming than those on interior points. The storage available in the last row, but not used, amounts to $N_v N(mp-M)/4$ segments. The latency occurs once in m rows; its value is approximately

$$L_{RL} = \frac{1/4 N_v N(mp-M)T_R}{m(\ell_n(1200t) + Tb)} \quad .$$

Similarly, for column-sweeping,

$$L_{CL} = \frac{1/4 N_V M(nq-N) T_R}{n(\ell_m(1200t) + T_b)} .$$

In order to determine overall internal latency, one must take account of the order of sweeping. For an equal number of row and column-sweeps,

$$L = L_D + 1/2(L_R + L_{RL} + L_C + L_{CL})$$

is a reasonably good measure of overall internal latency.

One latency term has not been included. It is the latency spent re-initializing between complete sweeps. There will be no attempt here to calculate it, although in some problems it may be important.

4.3 Storage Requirements

4.3.1 Fast Memory

If one examines the sequence of I/O for sweeping the mesh, he can tally all mesh blocks and edges contained in fast memory at every instant, and determine the amount of storage needed for the data as a function of time. If the amount of storage needed for program and scratch area is added, and the maximum over time of the total storage required is determined, one can state whether the scheme will work for a memory of given size. Alternatively, one may examine storage requirements over a sample of problems and problem sizes, and attempt to estimate the amount of fast memory required for a particular installation.

The maximum amount of fast memory required for storage of the mesh data is a function of the mesh and scheme parameters. It is also, in a somewhat odd way, a function of the organization of the fast memory itself, and of the size of the smallest addressable segment on disk and the length of the disk track.

The storage required for sweeping in transposed mode is greater than for sweeping in normal mode. The difference is of the order of an edge group, but it must be remembered that edge groups are usually larger if a transposed read is required, since storage for individual edges rather than edge groups is rounded upwards to the nearest disk segment.

Since there are many numerical PDE problems which do not require changing the direction of sweep, it is worthwhile to investigate the requirements for normal reading separately from transposed reading. In this report only normal reading will be analyzed.

Because of the parallel structure of ILLIAC IV, special problems arise in the allocation of memory. One must be clever in the design of the program and in the distribution of data across the processing elements. One of the constraints imposed in the analysis, that of considering 8×8 squares as the smallest subdivision of the mesh, resulted from taking account of the structure of the fast memory. This structure also causes problems with storage of edge values. If an edge or an edge group is packed tightly into the smallest number of quadrant words that will contain it, then it is likely that some of the edge points will not be located in the proper processing elements. Additional code will be needed to route data to proper PE's when the data is needed. The space saved by packing the edges or edge groups may be used up by the additional code and scratch areas.

Nevertheless, in this analysis we will calculate storage requirements based on having data packed moderately tightly. For row-normal reading an edge group consists of $8N_v kqd$ words. The number of disk segments needed to contain an edge group is

$$W_S^{EGR} = \left\lceil \frac{8N_v kqd + 255}{256} \right\rceil .$$

The number of quadrant words needed as an I/O area for an edge group is

$$W_{SQ}^{EGR} = 4W_S^{EGR} .$$

Likewise, the number of segments needed to contain a mesh block is

$$W_S^{MB} = \left\lceil \frac{N_v pq + 3}{4} \right\rceil ,$$

and the number of quadrant words needed as an I/O area for the mesh block is

$$W_{SQ}^{MB} = 4W_S^{MB} .$$

For calculations on block (i,j) , an area must be set aside to store the right edge of block $(i,j-1)$. As calculations on (i,j) sweep the block, old values from the block may be moved into the edge area so that when (i,j) is completely updated, the edge area contains the right edge

of (i,j) . The number of quadrant words needed for the single edge is

$$W_Q^{ER} = \left\lfloor \frac{8N_v pd + 63}{64} \right\rfloor .$$

We might include an 8×8 mesh square of storage as a token amount for overhead. This amounts to N_v quadrant words. If the I/O sequence for row-normal reading is examined, it can be seen that we need

$$W_{mem}^R = 3W_{SQ}^{EGR} + W_{SQ}^{MB} + \text{Max}(W_{SQ}^{EGR}, W_{SQ}^{MB}) + W_Q^{ER} + N_v$$

quadrant words of fast memory for storage of the data. The third term takes account of the case in which the ending of one row moves far into the beginning of the next. A similar expression W_{mem}^C exists for column-normal reading.

No attempt will be made here to estimate the storage required for program and scratch areas.

4.3.2 Disk

An easy way to manage disk is to allocate half of the disk to old mesh and edge groups and half to updated mesh and edge groups. This procedure takes no advantage, however, of the space on disk which becomes available for updated mesh blocks as the mesh is swept.

Referring once again to Figure 4, note that, starting from block $(1,1)$ for example, successive disk blocks are filled with each k blocks added to the mesh row. The last block in the mesh row is $(1, \ell_n^{k-2})$. If $\ell_n \leq T$, the mesh row will fit into one logical track, but if $\ell_n \geq T+1$, we will have to use another logical track for storage. The same situation

occurs for the other mesh rows. One mesh row would require $(\lfloor \frac{\ell_n}{T} \rfloor + 1)$ logical tracks.

It is now proposed that rather than using T out of T disk blocks for storage, we use only $T-1$ out of T blocks; so that we use another logical track for $\ell_n \geq T$, and yet another for $\ell_n \geq 2T$ (the second logical track is filled completely). In this way we insure that there will be an empty block immediately before blocks $(1,2)$, $(1,3)$, ..., $(1,k)$. We may then write updated blocks in these spaces as we sweep the row. Updated block $(1,k-2)$ is written in the space before $(1,k)$. $(1,k-1)$ is then written over $(1,1)$ since we do not need $(1,1)$ any more. Old blocks are thus successively overwritten by the updated $(k-2)$ th block following them in the row. One mesh row then requires $(\lfloor \frac{\ell_n}{T} \rfloor + 1)$ logical tracks; and m mesh rows require $m(\lfloor \frac{\ell_n}{T} \rfloor + 1)$ logical tracks.

Note that this procedure for managing disk also works for column-sweeping. Because of this we may choose the smallest of two possible storage requirements:

$$W_{LT}^{MB} = \text{Min}(m(\lfloor \frac{\ell_n}{T} \rfloor + 1), n(\lfloor \frac{\ell_m}{T} \rfloor + 1)) \quad .$$

No such game can be played with edge group storage. However, we may still choose the minimum requirement for two possible storage methods. Attention is drawn to the upper edge groups (superscript U) in Figure 4. One storage method is shown in the schematic. Groups with successive subscripts are stored in adjacent blocks on a logical track for $\ell_n \leq T$. Group $\bar{k}_T^U$ would be stored on logical track 3 in the first

position, although it is not shown since the drawing has $\ell_n = T-1$. Groups with successive Roman numerals simply wind around disk at intervals of T blocks, until the last one, $\overline{\ell_m k-2}$. For every increment of ℓ_m , another $(\lfloor \frac{\ell_n-1}{T} \rfloor + 1)$ logical tracks are added, as long as $\ell_m \leq 2T$. If one increases ℓ_m to $2T+1$, $2(\lfloor \frac{\ell_n-1}{T} \rfloor + 1)$ logical tracks must be added; however, we will consider only $\ell_m, \ell_n \leq 10$ and $T \geq 5$.

The second possible storage method is to put $\overline{R}_1, \overline{R}_2, \dots, \overline{R}_{\ell_n}$ on different tracks, and to put $I_1, \overline{k+1}_1, \dots, \overline{ik+1}_1$ in adjacent blocks on the same logical track for $i \leq T-1$. $I_1, II_1, III_1, \dots$ are still to be stored on the same logical track, spaced T blocks apart. The expression for the number of logical tracks required is the dual of the expression for the first method if $\ell_m, \ell_n \leq 2T$.

The number of logical tracks needed for upper edge groups is

$$W_{LT}^{EG} = \text{Min}(\ell_m (\lfloor \frac{\ell_n-1}{T} \rfloor + 1), \ell_n (\lfloor \frac{\ell_m-1}{T} \rfloor + 1)) \quad .$$

We allocate four such areas on disk, one each for old upper and lower edge groups and one each for the new.

The amount of disk storage needed is

$$W_{\text{disk}} = t(W_{LT}^{MB} + 4W_{LT}^{EG})$$

tracks. Note that this measure is in disk tracks and not logical tracks.

5. NUMERICAL SOLUTIONS

For a given problem M , N , N_v , d , T_K , a scheme must be found with a latency which is within an acceptable limit. The difference $T_R - T_K$ should be accounted for in the latency measure; however, this section will be an informal discussion of the existence of schemes for given values of M , N , N_v , d and will consider only the internal latency of the scheme.

If the values of M , N , N_v , T , t , k , ℓ_m , ℓ_n are specified, a scheme can be determined if all of the relations of the last section are satisfied. The latency may be calculated and tested for acceptability. The value T_R may also be calculated; as well as a maximum allowable value of d . If we are interested only in normal reading in either direction, then we may leave one of the ℓ -values unspecified, and impose some additional restriction or specify some other parameter.

A program has been written for the purpose of investigating the existence of solutions of the parameters for various problems M , N , N_v . A simple procedure is used: T , t , k , $\ell_{m/n}$ are iterated, and valid solutions for which the latency is less than or equal to .12 are printed. For each T , t , k values for ℓ_m and ℓ_n are tried successively in an attempt to find a scheme for column-normal or row-normal reading respectively. The program is written in Burroughs B5500 Extended Algol, and it is listed in the appendix to this report.

For each mesh size $M \times N$ squares, values 3 to 8 in unit steps were assigned to N_v . The free parameters T , t , k were iterated in unit steps over the ranges 5 and 6, 1 to 5, and 4 to 30 respectively. Values

for ℓ_m or ℓ_n were chosen from Table 2. The results of particular interest are the values of T_R and W_{mem} obtained. We would like to see many solutions, with values of T_R well distributed and with memory requirements very low. A survey of the results that have been obtained is presented in Table 4. All of the solutions have $W_{\text{mem}} \leq 1500$ and $W_{\text{disk}} \leq 48$. The smallest problem listed is core contained for $N_v = 3$ and 4. The largest problem listed represents about one-third of disk capacity for $N_v = 8$.

For each solution a maximum allowable value of d , d_{max} , is calculated. If $d_{\text{max}} < 3$, the solution is rejected. Tests have shown that no more than eight per cent of the solutions obtained by the program are rejected on this basis. Fast memory requirements are calculated for $d = 3$. There are few, if any, finite difference stencils in use for which d is greater than three; so that it is justifiable to group all solutions with $d_{\text{max}} \geq 3$ into one class. Each of these solutions is valid for $d \leq 3$.

The smallest and largest values of T_R , over the entire range of the parameter N_v , are listed. For the large meshes the solutions are numerous over small ranges of T_R ; and the trend seems to be that solutions are fewer for smaller meshes and occur over larger ranges of T_R . The distribution over T_R is discussed below. At this point a comment should be made concerning the method of finding the solutions.

The program used to obtain the results incorporates an artificial restriction on the mesh block size which is equivalent to an attempt to minimize the value of T_R for given scheme parameters. Within the program the parameters p and b are calculated; and from these a largest value of q is determined such that the bound $N_v p q \leq 4(b-2)$ is satisfied. If q were

<u>M × N</u>	<u>Number of Solutions</u>	<u>T_R Range</u>	Percent of Solutions With $W_{\text{mem}} \leq$		
			<u>400</u>	<u>800</u>	<u>1200</u>
20 × 20	0				
28 × 32	270	5.21 - 59.43	55	100	100
30 × 40	154	5.83 - 44.00	48	99	100
35 × 35	27	6.00 - 48.00	41	100	100
47 × 47	89	5.16 - 28.37	35	85	100
45 × 55	405	5.10 - 24.44	29	79	97
55 × 65	497	5.09 - 17.09	23	73	94
60 × 70	400	5.11 - 14.00	36	94	100
90 × 70	507	5.10 - 11.91	33	88	98
60 × 110	675	5.08 - 12.27	29	78	96
90 × 110	567	5.07 - 11.02	33	83	95

Table 4. Survey of results obtained on 11 mesh sizes.
Latency $\leq .12$, normal reading in either direction.

iterated downward from this largest value, solutions with larger T_R might be found. The program uses only the largest q ; the attempt is repeated with the variables p and q interchanged. There is another constraint on q which leads us to expect higher T_R for smaller mesh dimensions. Given n and N , in order to minimize internal latency, q should be the smallest value such that $qn \geq N$. For smaller mesh dimension N , this bound is more likely to be lower than the bound mentioned above. In fact, most of the solutions for the small meshes had m or n equal to one and p or q spanning an entire dimension of the mesh. In such solutions an entire column or row of 8×8 squares would be read at a time; and the W_{mem} were overestimated by the program because the edge group areas allocated would not be needed.

A rigorous mathematical investigation of the existence of solutions to the system of relations has not been performed; and the program used does not find every solution possible in the given T_R ranges. In fact, because of the peculiar behavior of the remainder terms in the integer divisions, the program may not even find the solutions with smallest T_R for the iterated parameters. The results obtained, however, are interesting even without the assurance that all possible solutions have been found.

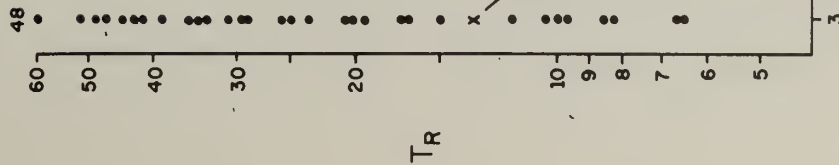
The last three columns of Table 4 give the percents of the solutions found for which fast memory requirements are less than or equal to 400, 800, and 1200 quadrant words. For the larger meshes, a larger percentage of solutions have high memory requirements. This does not necessarily indicate, however, that larger meshes require more memory. No attempt has been made at this time to examine in detail the fast memory requirements; but this problem is worthy of further investigation.

The relationship between T_R and N_V is illustrated in Figure 10 for four pairs of mesh dimensions. The highest and lowest values of T_R found are plotted for each value of N_V . In addition, selected solutions for N_V equal to 3 and 8 are plotted, and the number of solutions found is printed above the highest point for each N_V . The additional points at $N_V = 3$ and $N_V = 8$ are selected to represent the densities over T_R of the solutions obtained. It is seen that the density increases with increasing mesh size. Further tests are yet to be performed to determine whether the procedure of iterating p or q downward yields values of T_R which would close the gaps in the higher regions.

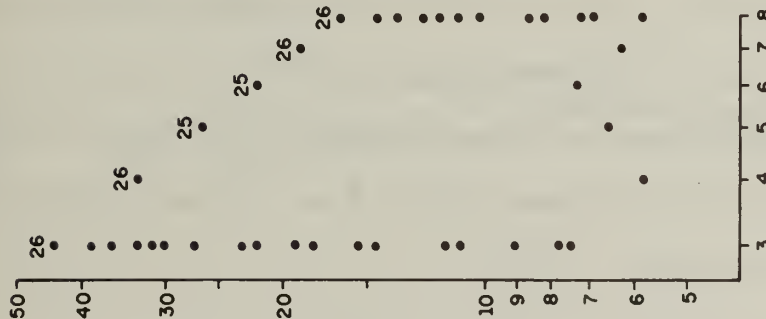
Note that the highest points for the two smallest meshes form straight lines. The six points in each of the graphs represent the same solution with only the difference that for smaller N_V the value of T_R is greater and the fast memory requirement is lower. A solution for the problem M, N, N_V, d, T_K is also valid for the problem $M, N, N_V-1, d, \frac{N_V}{N_V-1} T_K$; but it is not in general valid for the problem with N_V+1 variables because of the requirement $N_V pq \leq 4(b-2)$. The term "solution" here refers to the set of parameters $\{T, k, t, \ell_m, m, \ell_n, n, p, q, b, L, W_{\text{disk}}\}$ where L is the internal latency of the scheme. Note that L is independent of N_V and T_R .

A straight line (with the slope indicated) may be drawn to the left from every solution on the graph; solutions exist along these lines. One such line is drawn on the first graph to indicate the existence of a solution at the point marked "x". This solution would also be found using the procedure of iterating p or q downward. This procedure might also yield solutions for $N_V=3$ which are invalid for higher N_V .

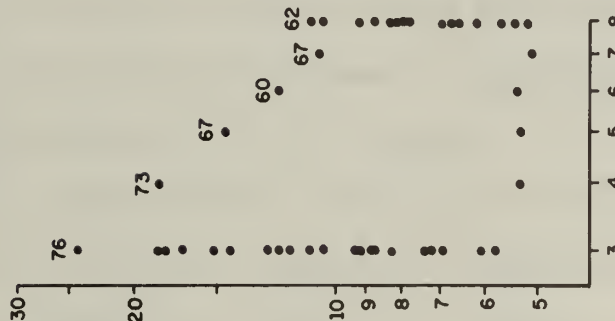
28 x 32



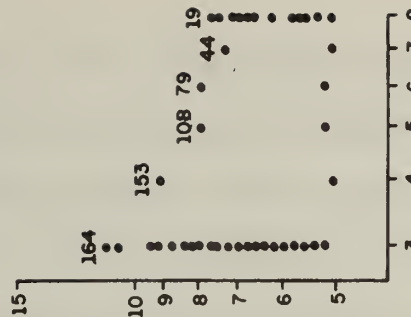
30 x 40



45 x 55



90 x 110



N_V

Figure 10. T_R vs. N_V for the solutions found for four mesh sizes. The number of solutions is printed for each value of N_V .

The most interesting region of the graph is the low end of the T_R scale, since for smaller T_R the disk more closely approximates a fast memory for the restricted class of problems considered here. The results in this region are better for the larger meshes than for the smaller meshes in both the density of solutions and in the minimum T_R obtainable. Figure 11 shows the minimum values on the same graph.

A fact of moderate interest is that a solution for the problem M, N, sN_v, d, T_K where s is an integer can be modified to accommodate the problems sM, N, N_v, d, T_K or M, sN, N_v, d, T_K independently of the direction of sweeping. The modification consists of multiplying p or q by s ; T_R is unchanged and W_{mem} decreases. Non-integer values may be substituted for s if one is careful to insure that wherever s is used as a multiplier, the result is an integer.

In an automatic method for finding the optimum solution for a given problem, the external latency should be added to the internal latency, and the result could be used as a measure of the total latency. The solution with the smallest total latency would be the optimum solution. The external latency is given by

$$L_{\text{external}}^C = \frac{\frac{M}{p}(T_R - T_K) \frac{N_v pq}{4}}{\ell_m(1200t) + Tb}$$

for column sweeping. An exact expression would be obtained by replacing Tb in the denominator by $T \frac{1200t}{kT-1}$. The search for a solution could continue over a wide range of the parameter T . Real values of $T \leq T_K$ or even slightly greater than T_K might be tried. Of course the interval

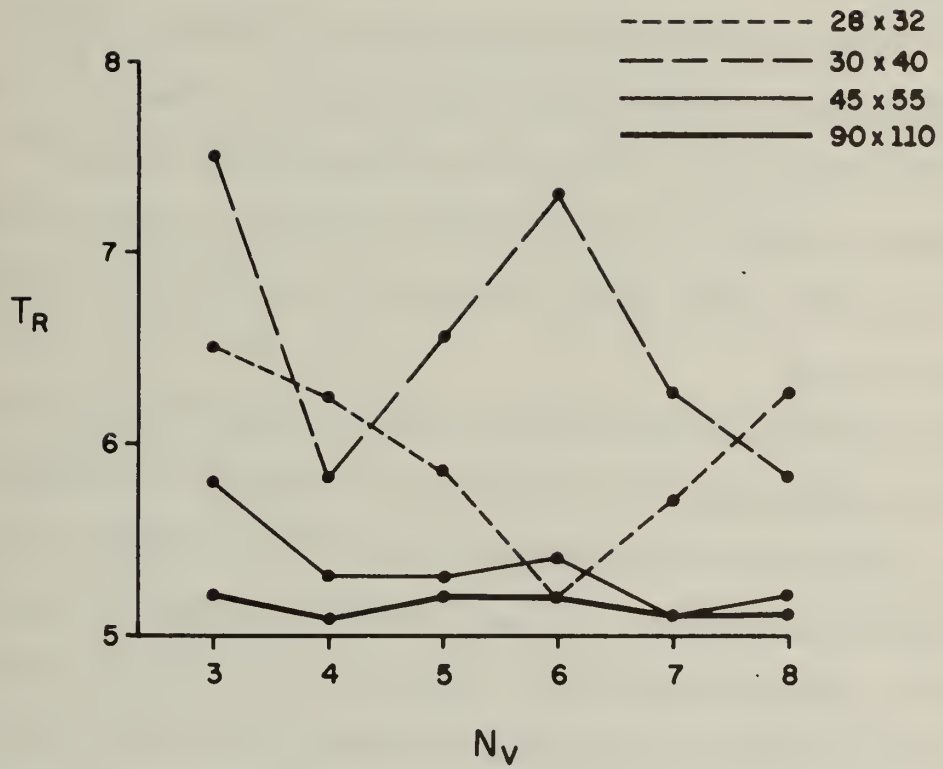


Figure 11. Minimum values of T_R found.

over which T is iterated should be properly discretized to assure that disk blocks do not begin within a disk segment.

A search over very many values of T would be lengthy, and an automatic method should incorporate a procedure to decide when to stop. The first solution found with an acceptable total latency might be taken, for example.

The efficiency of disk allocation (i.e., the percentage of allocated space which is actually used) has not been considered here, but it should be noted that for $T \ll T_R$ this efficiency cannot be high. If this is an important factor, one might start the search with higher values of T . One of the ℓ_m or ℓ_n would have to be close to a multiple of T for high storage efficiency.

We can expect that solutions would be fewer and memory requirements higher for problems requiring changes in the direction of sweeping. Changing direction might be useful in handling boundaries of the mesh, for example. It is meaningless, however, to speak of direction changes for solutions with p or q spanning a dimension of the mesh; there is only one direction possible. Since most of the solutions for small meshes were of this type, we might expect that changing direction could be meaningful only for large meshes.

It seems that the ability to change direction of sweep is of questionable importance. In this light the results given would indicate that the scheme presented in this report can be useful for non-core-contained problems.

6. CONCLUSION

The scheme described in this paper is only one of a family of schemes that could work. Another member of the family is obtained by interchanging the positions of upper and lower edge groups $\bar{R}_i^U$ and $\bar{R}_i^L$ and shifting all of the mesh blocks in Figure 4 left by T disk blocks.

It might be possible to find a scheme which would work with a logical period T of four. This might be done by allowing transmissions of two edge groups within the space of one disk block. It is clear, however, that no scheme could have a logical period T or a real period T_R less than two, since every mesh block must be input and output. The value two is an obvious lower limit on the period.

There are a number of problems in which several different kernels are to be applied to different subsets of the N_V variables. There is no provision in the present scheme for handling these types of problems efficiently; on each sweep of the mesh, all of the variables are transmitted. If one transmits only the variables needed, then the value of T_R is effectively increased since the logical period T must remain constant during problem execution. Remapping of the mesh onto disk might be considered, but this would not be an easy procedure to use. The two problems of changing the period and handling overlapping subsets of the N_V variables are somewhat related and are worthy of further investigation.

APPENDIX

The following is a listing of the Burroughs B5500 Extended Algol program used to obtain the numerical results. A note may be helpful to the reader who is unfamiliar with the language. The operator "/" results in a floating point division with a floating point result. The operator "DIV" indicates a fixed point division with truncation of the remainder. The expression "A DIV B" is equivalent to $\lfloor \frac{A}{B} \rfloor$.

```

BEGIN
REAL ARRAY PCT[1:20];
REAL TREAL,RULT,ITCY,CP,DM,DN,LTCYM,LTCYN,TR,TRMIN,TRMAX;
INTEGER ARRAY TH[5:12,0:7],HIST[1:20],MMM,NNN[1:15];
INTEGER NV,TP,DMAX,K,1,P,Q,NVPQ,B,          KTM1,LT,RM2,FBM2, UNFIL,
          N,M,NN,MM,LM,LN,ILM,ILN,S,      SUM,MAXPQ,      I,DSKM,OSKN,
          EDGM,EDGN,DISK,LMKTM1,LNKTML,ISZ,TI,D,ENPK,WEGSQ,WBBSQ,WFO,WMEM;
ALPHA SYM,SYM1;
FILE PTOUT 15(1,1/);
LABEL SKIP2,SKIP2,CYC;
LIST LST(NV,TP,LM,MM,(N,N,TREAL,SYM,      D,K,T,P,Q,NVPQ,B,          UNFIL,RULT,
          LTCY,SYM,SYM1,      DM,DN,WMEM,DISK));
FORMAT FA(//215,"      = MM,NN      CRITERION =" ,F5.3/
          "NV TP LM M LN N TREAL D K T P Q NVP
Q      B UNFIL RULT LTCY      DMRL DNRL MEMRY DISK"),
          FR(12,X2,12,2*(X3,12,14),X2,F5.2,X1,A1,X2,12,X3,12,X3,11,15,14,
          X3,14,X3,13,X3,13,X2,F4.3,X4,F4.3,X1,2A1,X4,X3,2F5.1,18,15),
          FNV("***** TRMIN="F5.2" TRMAX="F5.2" SUM="15),
          FHSTD("FAST-MEM REQUIREMENT DISTRIBUTION"X5"NO. OF SOLNS ="16,
          X10"FOR"15"X"13//
          " Q-W"X6"REQ PCT ACCUM"/),
          FHST(14,110,F7,1);
FILL TH[5,*] WITH 4,1,5,6,10;
FILL TH[6,*] WITH 5,1,3,6,7,9;
FILL TH[7,*] WITH 5,1,3,7,8,10;
FILL TH[8,*] WITH 5,1,3,5,8,9;
FILL TH[9,*] WITH 6,1,3,5,6,9,10;
FILL TH[10,*] WITH 6,1,3,5,6,7,10;
FILL TH[11,*] WITH 6,1,3,5,6,7,8;
FILL TH[12,*] WITH 7,1,3,5,6,7,8,9;
FILL MMM[*] WITH 30,20,28,35,45,47,55,60,60,90,90;
FILL NNN[*] WITH 40,20,32,35,55,47,65,70,110,70,110;
CH+.12;
FOR ISZ+6 STEP 1 UNTIL 11 DO
BEGIN
FOR I+1 STEP 1 UNTIL 20 DO HIST[I]+0;
WRITE(PTOUT,FA,PAGE);
MM+MMM[ISZ]; NN+NNN[ISZ];
WRITE(PTOUT,FA,MM,NN,CR);
FOR NV+3 STEP 1 UNTIL 8 DO
BEGIN
IF NV*MM*NN>210000 THEN GO TO SKIP2;
TRMIN+999;
TRMAX+0;
SUM+0;
FOR TP+5,6 DO
FOR T+1 STEP 1 UNTIL 5 DO
FOR K+4 STEP 1 UNTIL 30 DO
FOR ILM+1 STEP 1 UNTIL TH[TP,0] DO
BEGIN
MM+MMM[ISZ]; NN+NNN[ISZ];
SYM1+" ";
KTM1+K*TP-1;
B+(LT+1200*T) DIV KTM1;
UNFIL+LT-KTM1*B;
IF T=1 THEN SYM+" "
ELSE IF 1200 MOD B =0 THEN SYM+"E"

```

```

ELSE IF 2*(T-1)SUNFIL THEN SYM+"A"
ELSE BEGIN
    B+B-1;
    UNFIL+UNFIL+KTM1;
    SYM+"B"
END;

BM2+B-2;
FRM2+4*BM2;
LM+TRL[TP,ILM1;
M+LM*K-2;
CYC;
P+(MM+4-1) DIV M;
Q+FRM2 DIV (NV*P);
IF Q=0 THEN GO TO SKIP;
N+(NN+Q-1) DIV Q;
Q+(NN+N-1) DIV N;
LN+(N+1+K) DIV K;
NVPQ+NV*P*Q;
RIILT+IINFIL/LT;
DM+M-MM/P;
DN+N-NV/Q;
TREAL+4*TP*B/NVPQ;
LTCYM+((TP*(3+DM)-LM)*B+.25*NV*MM*(Q-NN/N)*TREAL)/
    (LM*LT+TP*B);
LTCY+LTCYM+RIILT;
IF LTCY>CR THEN GO TO SKIP;
DMAX+(256*BM2) DIV (ENPK+8*NV*P*K);
IF DMAX<3 THEN GO TO SKIP;
D+3;
WFGSQ+4*((D*ENPK+255) DIV 256);
WMBSQ+4*((NVPQ+3) DIV 4);
WFQ+(8*NV*Q*D+63) DIV 64;
WMEM+3*WFGSQ+WMBSQ+WEQ+NV*(IF WFGSQ>WMBSQ THEN WFGSQ
    ELSE WMBSQ);
IF WMEM>1500 THEN GO TO SKIP;
S+((IF WMEM<2000 THEN WMEM ELSE 2000)-1) DIV 100 + 1;
DSKM+(4-ENTIER(DM))*(LN DIV TP + 1);
DSKN+N*(LM DIV TP + 1);
EDGM+LM*(LN-1) DIV TP + 1;
EDGN+LN*(LM-1) DIV TP + 1;
DISK+T*((IF DSKM<DSKN THEN DSKM ELSE DSKN)+
    4*(IF EDGM<EDGN THEN EDGM ELSE EDGN));
IF DISK>48 THEN GO TO SKIP;
IF SYM1="N" THEN
    BEGIN
        TI+LM; LM+LN; LN+TI;
        TI+N; N+1; N+TI;
        TI+P; P+Q; Q+TI;
        IR+DM; DM+DN; DN+TR
    END;
WRITE(PTOHT,FR,LST);
IF TRMIN>TREAL THEN TRMIN+TREAL;
IF TRMAX<TREAL THEN TRMAX+TREAL;
SUM+SUM+1;
HIST(S)+HIST(S)+1;
SKIP: IF SYM1=" " THEN
    BEGIN

```

```

MM+MNN[ISZ]); NN+MMM[ISZ]);
SYM1+"N";
GO TO CYC
END
END;
WRITE(PIOUT,FNV,TRMIN,TRMAX,SUM)
END;
WRITE(PIOUT[PAGE1]);
SUM+0;
FOR I+1 STEP 1 UNTIL 20 DO PCT[I]+SUM+SUM+HIST[I];
WRITE(PIOUT,FHST0,SUM,MMM[ISZ],NNN[ISZ]);
IF SUM>0 THEN
FOR I+1 STEP 1 UNTIL 20 DO
BEGIN
PCT[I]+PCT[I]/SUM;
WRITE(PIOUT,FHST,100×I,HIST[I],100×PCT[I])
END;
SKIP2;
END
END.

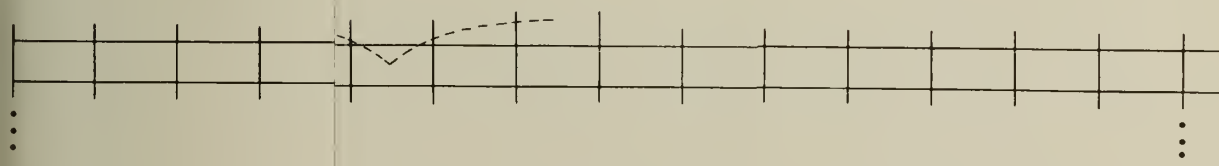
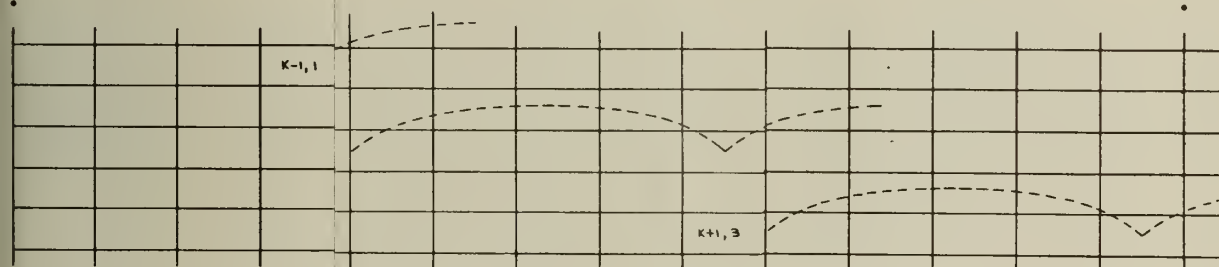
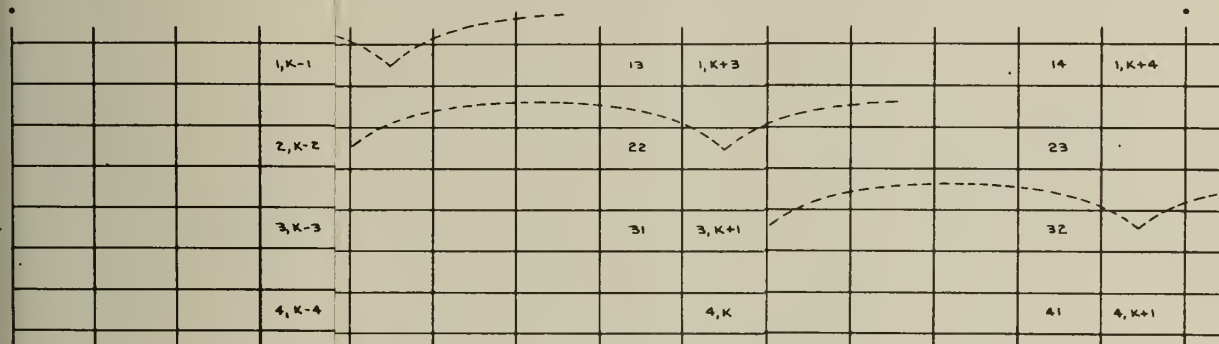
```

LIST OF REFERENCES

- [1] Barnes, G. H., et al., "The ILLIAC IV Computer," IEEE Transactions on Computers, Vol. C-17, No. 8 (August, 1968), pp. 746-757.

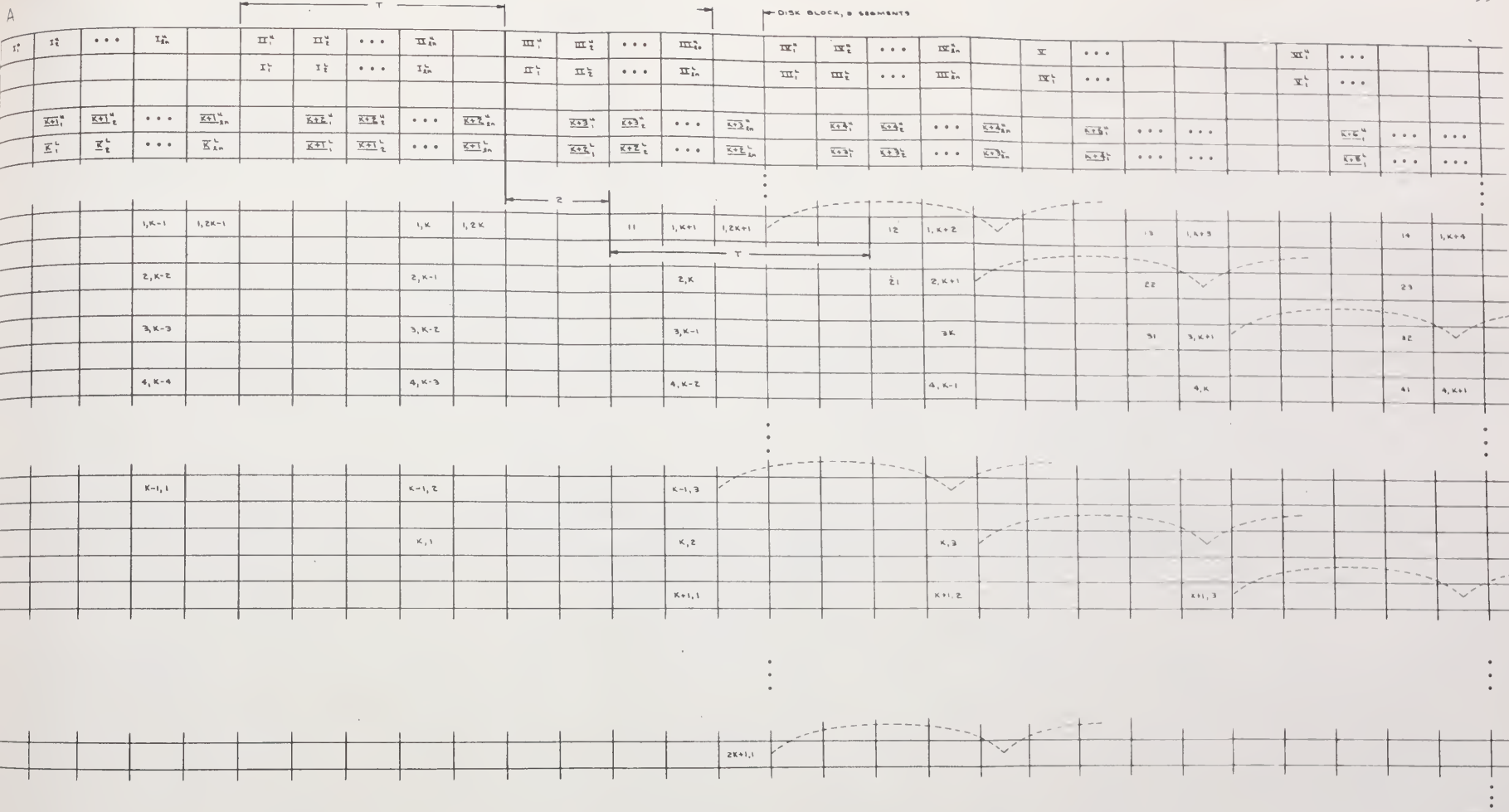
A

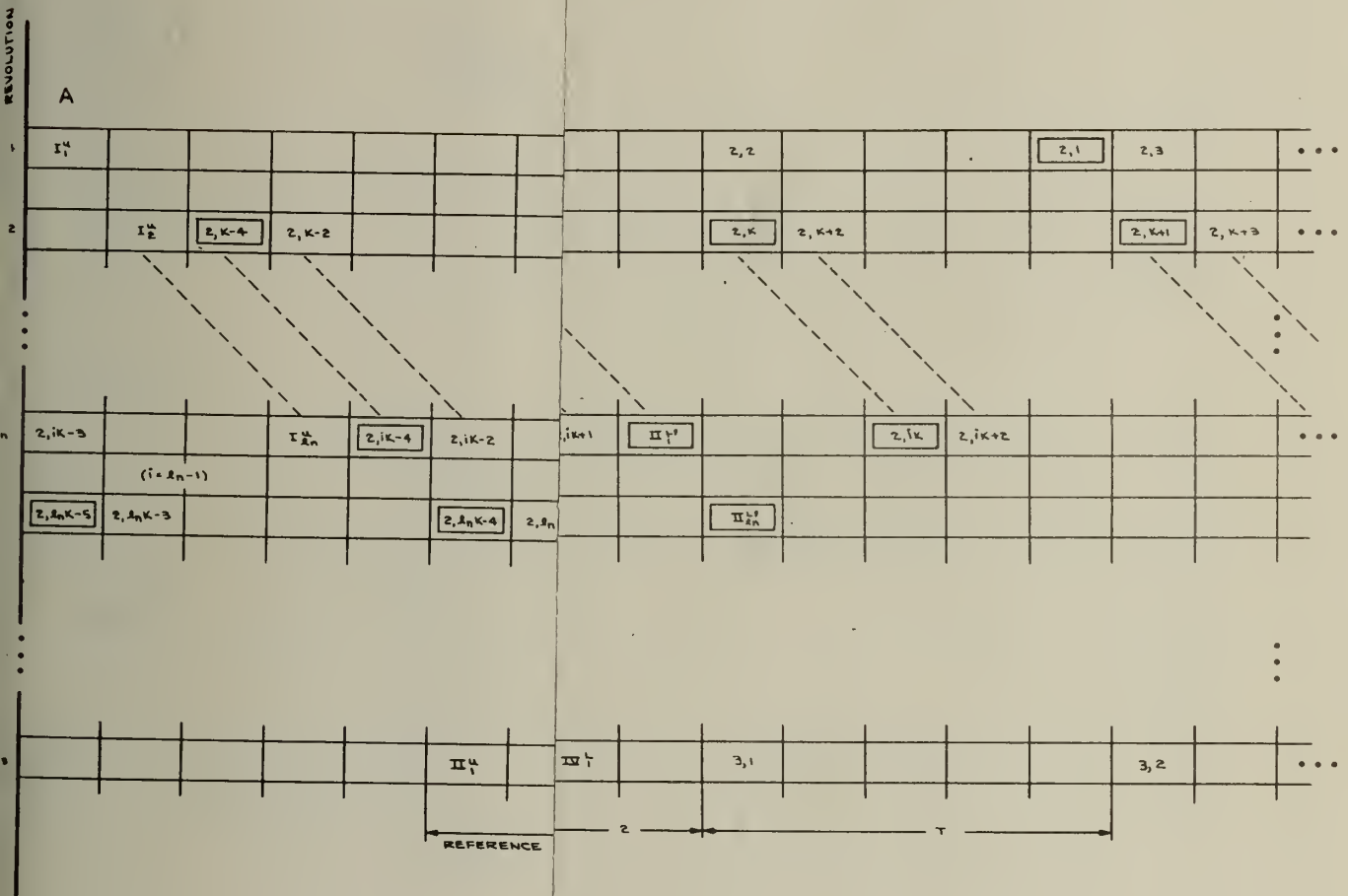
I_1^u	I_2^u	...	I_{2n}^u		Y	...				Y_1^u	...			
					Y_1^l	...				Y_1^l	...			
	$K+1_1^u$	$K+1_2^u$	...	$K+2_{2n}^u$		$K+5_1^u$	...	...		$K+6_1^u$	...	...		
	K_1^l	K_2^l	...	$K+3_{2n}^l$		$K+4_1^l$	...	...		$K+5_1^l$	...	...		



mission is started. This access time is, in general, not predictable; but it is bounded by the disk rotation time.

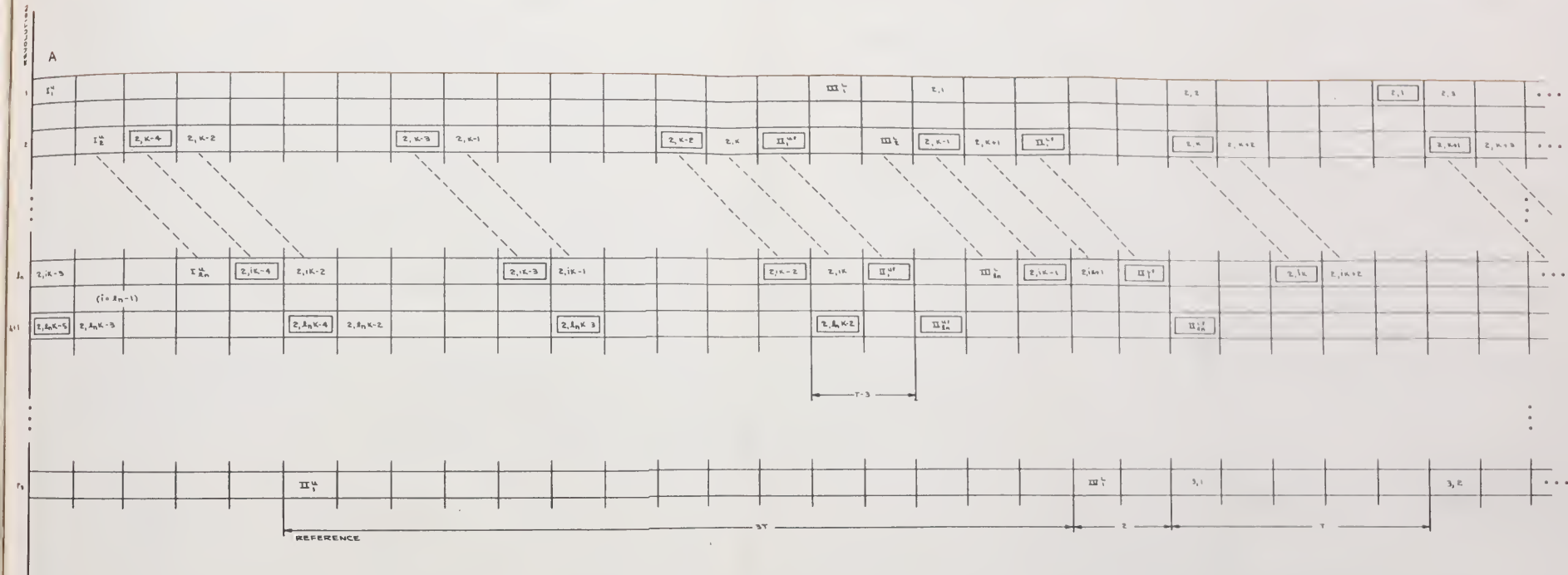
The access time during which the computer does useless work is latency; and the object of the present investigation to minimize the latency for a reasonably large class of problems.





mission is started. This access time is, in general, not predictable; but it is bounded by the disk rotation time.

The access time during which the computer does useless work is latency; and the object of the present investigation to minimize the latency for a reasonably large class of problems.



A

STARTING COLUMN 2

(I_1^{RT})					(II_1^{RT}) (IX_1^{RT})	(II_1^{RT}) (IX_1^{RT})	2,2			(II_1^{RT}) (IX_1^{RT})	1,2	3,2	...
$(K+1)_1^{RT}$ (K_1^{RT})	K-4,2	K-2,2				$(K+K)_1^{RT}$ $(K+K)_1^{RT}$	K,2	K+2,2		$(K+K)_1^{RT}$ $(K+K)_1^{RT}$	K+1,2	K+3,2	...

⋮

STARTING COLUMN K-2

(I_1^{RT})					(II_1^{RT}) (IX_1^{RT})	(II_1^{RT}) (IX_1^{RT})				(II_1^{RT}) (IX_1^{RT})			...
$(K+1)_1^{RT}$ (K_1^{RT})		2,K-2				$(K+K)_1^{RT}$ $(K+K)_1^{RT}$	4,K-2	6,K-2		$(K+K)_1^{RT}$ $(K+K)_1^{RT}$	5,K+2	7,K-2	...
$(K+1)_1^{RT}$ (K_1^{RT})		$(K+K)_1^{RT}$ (K_1^{RT})	K,K-2	K+2,K-2									

⋮

mission is started. This access time is, in general, not predictable; but it is bounded by the disk rotation time.

The access time during which the computer does useless work is latency; and the object of the present investigation to minimize the latency for a reasonably large class of problems.

A

STARTING COLUMN K-1

(π_1^{RT})					(π_1^{RT})				(π_1^{RT})				...
	$(\overline{K+1}_1^{RT})$			$1, K-1$		$(\overline{K+1}_1^{RT})$ $(\overline{K+1}_1^{RT})$	$3, K-1$	$3, K-1$		$(\overline{K+1}_1^{RT})$ $(\overline{K+1}_1^{RT})$	$4, K-1$	$4, K-1$	...
$(\overline{K-1}_1^{RT})$		$(\overline{K+1}_1^{RT})$ $(\overline{K+1}_1^{RT})$	$K-1, K-1$	$K+1, K-1$	$(\overline{K-1}_1^{RT})$								...

...

STARTING COLUMN K

	(1^{RT})					$(\overline{K-1}^{RT})$ $(\overline{K-1}^{RT})$	$2, K$	$4, K$		$(\overline{K-1}^{RT})$ $(\overline{K-1}^{RT})$	$3, K$	$3, K$	...
		$(\overline{K-1}^{RT})$ $(\overline{K-1}^{RT})$	$K-2, K$	K, K	$\overline{K-1}^{RT}$								...

when the system must wait for nearly a complete disk rotation before the transmission is started. This access time is, in general, not predictable; but it is bounded by the disk rotation time.

The access time during which the computer does useless work is latency; and the object of the present investigation to minimize the latency for a reasonably large class of problems.

STARTING COLUMN K-1

•

STARTING COLUMN K

[illegible]

Security Classification

UNCLASSIFIED

Security Classification

DOCUMENT CONTROL DATA - R & D

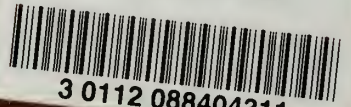
(Security classification of title, body of abstract and indexing annotation must be entered when the overall report is classified)

1. ORIGINATING ACTIVITY (Corporate author) Department of Computer Science University of Illinois at Urbana-Champaign Urbana, Illinois 61801		2a. REPORT SECURITY CLASSIFICATION UNCLASSIFIED	
		2b. GROUP	
3. REPORT TITLE DISK I/O FOR NON-CORE-CONTAINED P.D.E. MESHES AND ARRAYS			
4. DESCRIPTIVE NOTES (Type of report and inclusive dates) Research Report			
5. AUTHOR(S) (First name, middle initial, last name) Bruce Allen Bernott			
6. REPORT DATE March 5, 1969		7a. TOTAL NO. OF PAGES 67	7b. NO. OF REFS 1
8a. CONTRACT OR GRANT NO. 46-26-15-305		9a. ORIGINATOR'S REPORT NUMBER(S) DCS Report No. 311	
b. PROJECT NO. USAF 30(602)4144			
c.		9b. OTHER REPORT NO(S) (Any other numbers that may be assigned this report)	
d.			
10. DISTRIBUTION STATEMENT Qualified requesters may obtain copies of this report from DCS.			
11. SUPPLEMENTARY NOTES NONE		12. SPONSORING MILITARY ACTIVITY Rome Air Development Center Griffiss Air Force Base Rome, New York 13440	
13. ABSTRACT In finding discrete solutions of systems of partial differential equations on a computer, one is faced with the problem that the desired number of mesh points may exceed the machine's fast memory. This problem will be common on machines such as ILLIAC IV, for which extremely high computing power invites the use of meshes many millions of words in size. Because of the high dollar price of fast memory, it is sensible to look at large disk stores with high transmission speeds as back-up storage for meshes and arrays. The main problem encountered is the access time of such a storage unit. The address of a block of data stored on disk might be taken as the address of the first word in the block. This address must specify both track number and radial position of the block. If the computer issues a command to transmit the block immediately after the beginning of the block has passed the reading head, then the system must wait for nearly a complete disk rotation before the transmission is started. This access time is, in general, not predictable; but it is bounded by the disk rotation time. The access time during which the computer does useless work is latency; and the object of the present investigation to minimize the latency for a reasonably large class of problems.			

14	KEY WORDS	LINK A		LINK B		LINK C	
		ROLE	WT	ROLE	WT	ROLE	WT
	partial differential equations mesh arrays back-up storage disk rotation time reading head discrete solutions						

FEB 7 - 1974

UNIVERSITY OF ILLINOIS-URBANA
510.84 IL6R v.3 C002 v.308-315(1969)
TRANQUIL : a language for an arrey proce



3 0112 088404311